Program Models and Semi-Public Environments

D. Grossi[1], W. van der Hoek[1], C. Moyzes[1], and M. Wooldridge[2]

[1]University of Liverpool, UK
[2]University of Oxford, UK

March 30, 2016

Abstract

We develop a logic for reasoning about *semi-public environments*, that is, environments in which a process is executing, and where agents in the environment have partial and potentially different views of the process. Previous work on this problem illustrated that it was problematic to obtain both an adequate semantic model and a language for reasoning about semi-public environments. We here use *program models* for representing the changes that occur during the execution of a program. These models serve both as syntactic objects and as semantic models, and are a modification of action models in Dynamic Epistemic Logic, in the sense that they allow for ontic change (i.e., change in the world or state). We show how program models can elegantly capture a notion of observation of the environment. The use of these models resolves several difficulties identified in earlier work, and admit a much simpler treatment than was possible in previous work on semi-public environments.

1 Introduction

We are interested in settings where agents are involved in a (possibly non-deterministic) computational activity in which the *computation* itself is performed publicly, but the *data* is distributed privately. Specifically, agents are assumed to have only partial knowledge of the values of the variables used in the computation. In this setting, the basic question to which we address ourselves in this work is: *What can we say about what the agents know and learn as the computation proceeds?*

We begin with a simple example, to illustrate the general setting of semi-public environments:

> Suppose we have three variables, say x, y, and z, and we have an agent i who can see *only* the variable z. We aim to have a program that swaps the values of x and y, using only assignments of the form $a := b$, where a and b are program variables. Now, *can we do this*

> *in our setting without i learning anything about the values of the variables x and y?* Although i cannot directly see the value of x or y, the "standard" solution to this problem (using z as a temporary variable) will result in i knowing the value of one of the variables.

Van der Hoek *et al* [22] give more examples: among others, they show how the *dining cryptographers* problem can be modelled as a semi-public environment. Again, this is an example of a setting where there is a commonly agreed computation, but the data (the outcomes of the toss of a coin) is not.

Although there is much related work in the literature, to the best of our knowledge this issue was first explicitly considered in [22]. This work illustrates that it is problematic both to develop an appropriate language for reasoning about semi-public environments, and at the same time develop an appropriate semantic model for such a language. The approach presented in the current paper resolves several difficulties and limitations identified by Van der Hoek *et al* [22] (see below, when describing our contributions).

Our work in particular, and semi-public environments in general, are in the intersection of at least three closely related research areas:

- First, they are closely related to *interpreted systems* [8], where what an agent knows derives from what agents can see about their environments (typically, their internal state).
- Second, the way in which we model programs in our work is close to that of Propositional Dynamic Logic (DL, [12]), which uses basic programs as the building blocks, and combines them to create more complex ones. Note that PDL is a language for reasoning about *ontic change*, i.e., change in the real world or the real state.
- Third, our work is a close relative of Dynamic Epistemic Logic (DEL, [27]), which is intended for reasoning about *information change*, i.e., changes in the knowledge or beliefs of agents.

One approach to modelling informational change in DEL is with using *action models* – see [2, 3] and [27, Chapter 6]. Their domain is that of abstract actions together with an accessibility relation capturing which actions 'look the same' for which agents. The first papers on action models already suggested that it would be 'obvious' to include mechanisms to deal with ontic change in such models. Work in DEL that tried to study ontic change and informational change in one and the same framework were subsequently undertaken in [28, 29], in [21] and in [25], among others.

We utilize such models, where actions can have both an ontic (change the world) and an epistemic (change the information) effect to build *program models* where the notion of visibility of the environment plays a central role.[1]

[1] We use the term *program models* for models that represent both ontic and epistemic change, and *action models* for actions modelling epistemic change. Note that [2] uses the term program models for what we call action models.

A key design principle in this paper is that of simplicity: we make a number of assumptions that keep our approach technically lightweight. For example, programs handle propositional variables, and there are only finitely many such variables. In addition, we assume that which agent sees which variables is common knowledge. Finally, we allow public announcements only on objective formulas, rather than on epistemic/dynamic ones. Obviously, lifting (some of) these assumptions offers interesting questions for further research.

Our paper makes the following contributions: we give a realistic interpretation of the program construct $\cup$ in semi-public environments, different from that of [22]; we employ program models to give a clear interpretation to such programs; we show that the logic for semi-public environments is in fact equivalent to epistemic logic enriched with a notion of *vision*; our use of program models is the first to employ action models that give an account on such a notion; using program models, we obtain a relatively simple and natural axiomatisation for semi-public environments, without the need to use a universal modality, as in [22] and, finally, in contrast with the mentioned paper, we are able to shed light on sequential composition and iteration within the context of semi-public environments.

The remainder of the paper is structured as follows:

- In Section 2 we provide the context, language, and models of our logic along with truth definitions for its formulas.
- Section 3 contains the definition for the class of program models, and the specific program models corresponding to the several meta-logical programs. We then show that these two classes of models coincide. Section 3 also contains a finite axiomatisation and a completeness result – we point out the validities that will serve as axioms along the way. Section 4 refers to notions of equality between our programs.
- Section 5 discusses related work and concludes.

2 Preliminaries

Throughout the paper, we will be dealing with a set $Ag = \{1, \ldots, m\}$ of agents, and a set $\mathcal{V}ar = \{x_1, \ldots, x_n\}$ of propositional variables. The variables $\mathcal{V}ar$ will be those manipulated by the programs. For each agent $i \in Ag$, we let $V(i) \subseteq \mathcal{V}ar$ be the set of variables visible to i. Thus the set $V(i)$ represents i's view of the environment. The set of propositional formulas is denoted by $\mathcal{L}_0$. A valuation $\theta : \mathcal{V}ar \to \{true, false\}$ assigns a truth value to each variable $x_j \in \mathcal{V}ar$. The set of all valuations is denoted by Θ. For two valuations θ_1, θ_2, we write $\theta_1 \sim_i \theta_2$ if for all $x \in V(i)$, $\theta_1(x) = \theta_2(x)$, i.e., θ_1 and θ_2 coincide on the values of the variables observed by agent i. We extend this definition in a natural way; for $I \subseteq Ag$ we write $\theta_1 \sim_I \theta_2$ if for all $i \in I$, $\theta_1 \sim_i \theta_2$.

Definition 1 (Toggling values) *Let $\theta \in \Theta$ and $S \subseteq \mathcal{V}ar$. The toggling of*

values for the variables in S in θ, notation $\updownarrow(S)(\theta)$, is defined as follows:

$$\updownarrow(S)(\theta)(x) = \begin{cases} \textit{not } \theta(x) & \textit{if } x \in S \\ \theta(x) & \textit{otherwise} \end{cases}$$

Also, slightly abusing notation, for $\varphi_0 \in \mathcal{L}_0$ we define $\updownarrow(S)(\varphi_0)$, as:

$$\updownarrow(S)(\varphi_0) = \begin{cases} \varphi_0 & \textit{if } S = \emptyset \\ \updownarrow(S \setminus \{x\})\varphi_0[\neg x/x] & \textit{for any } x \in S \end{cases}$$

where $\varphi_0[\neg x/x]$ denotes the formula φ_0 with every occurrence of x replaced by $\neg x$.

Given two sets of variables S_1 and S_2, we denote $S_1 \triangle S_2$ to be their symmetric difference, i.e., $S_1 \triangle S_2 = (S_1 \cup S_2) \setminus (S_1 \cap S_2)$.

So $\updownarrow(V)(\theta)$ is like θ, but with the assignment of values to variables in V toggled. Obviously, $\updownarrow(\emptyset)$ is the identity function: $\updownarrow(\emptyset)(\theta) = \theta$. An examples of applying a toggle to a formula is obtained by $\updownarrow(\{x, y\})((x \vee \neg y) \leftrightarrow z)$ which yields $(\neg x \vee \neg\neg y) \leftrightarrow z$.

2.1 Language and Models

We now define our object language for reasoning about semi-public environments. In our setting, programs have both a syntactic and semantic flavour: our basic construct is a *pointed program model*, denoted (M, w) or M, w, where w is called a *program point*. The set of pointed program models will be formally defined in Section 3; we denote this set by PM. The fact that a program has several program points is to cater for the uncertainty by the agents in the environment as to what the exact changes are, enforced by the program. The language is an extension of the well-known multi-agent epistemic language S5$_n$ [8]. We have modal epistemic operators K_i, where $K_i\varphi$ expresses that agent i knows φ, an operator V_i, where $V_i x$ indicates that variable x is visible to agent i, and a dynamic "box" operator $[\mathsf{M}, \mathsf{w}]$, where $[\mathsf{M}, \mathsf{w}]\varphi$ means that the execution of M, w leads to states where φ holds. Formally, the syntax of formulas φ of the language $\mathcal{L}$ is defined by the following grammar:

$$\varphi \ ::= \ \top \mid x_j \mid V_i x_j \mid \neg\varphi \mid \varphi \wedge \varphi \mid [\mathsf{M}, \mathsf{w}]\varphi \mid K_i\varphi$$

where $i \in Ag$, $x_j \in \mathcal{V}ar$, and $\mathsf{M}, \mathsf{w} \in \mathsf{PM}$. We will use the standard classical logic abbreviations for $\bot, \vee, \rightarrow$ and $\leftrightarrow$, and we write $M_i\varphi \stackrel{\circ}{=} \neg K_i \neg\varphi$ and $\langle \mathsf{M}, \mathsf{w}\rangle\varphi \stackrel{\circ}{=} \neg[\mathsf{M}, \mathsf{w}]\neg\varphi$. We also introduce $[\mathsf{M}]\varphi$ for $\bigwedge_{\mathsf{w} \in \mathsf{M}} [\mathsf{M}, \mathsf{w}]\varphi$. A formula $\psi \in \mathcal{L}$ is called *program-free* if it has no occurrences of $[\mathsf{M}, \mathsf{w}]$. Recall that the propositional fragment of $\mathcal{L}$ is denoted by $\mathcal{L}_0$.

Definition 2 (Epistemic Models) *An* epistemic model *M for $\mathcal{L}$ is a tuple*

$$M = \langle W, R, V, f\rangle, \ \textit{where}$$

1. *W is a finite (possibly empty) set of states;*
2. *$f : W \to \Theta$ assigns a valuation θ to each state in W;*
3. *$V : Ag \to 2^{\mathcal{V}ar}$ keeps track of the variables that agent i 'can see';*
4. *$R : Ag \to 2^{(W \times W)}$ assigns an accessibility relation to each agent $i \in Ag$. We write uR_iv or $R_i(u, v)$ rather than $(u, v) \in R(i)$. Each $R(i)$ is an equivalence relation, and moreover, we assume that uR_iv implies $f(u) \sim_i f(v)$.*

If $M = \langle W, R, V, f\rangle$, we write $w \in M$ for $w \in W$. For $u, v \in M$, we write $u \sim_i v$ if $f(u) \sim_i f(v)$. A pair (M, v) (with $v \in M$) is called a pointed model. Let $\mathcal{EM}_m$ denote the class of pointed epistemic models for m agents. Finally, the set of all vision functions V is denoted by $\mathcal{V}is$.

Note that R_i determines what agent i knows, while $\sim_i$ is the indistinguishability based on what i sees. We have $R_i \subseteq \sim_i$: i.e., an agent can know more than based on what he sees. Take for instance an agent i who sees x, and consider a program that first assigns x the value of y, and then assigns x the value of z. Although i does not see y, he does know its value after execution of this program, since this value was assigned to x and is not changed since (the formal definitions of these programs are given in Section 3).

We also ought to explain the definition of the set of states W. Given that the sets Ag and $\mathcal{V}ar$ are finite, and the set of possible values for the variables is finite (*true* or *false*), we can assume that $\mathrm{S5}_n$ is characterised by the class of finite epistemic models (cf. [15], Section 2.1.5, finite model property). Furthermore, as we will see later, program models are also finite, and thus there is no way an infinite model will occur after the execution of a program. Regarding the possibility of an empty model: it has no impact on our results (in terms of validities, for instance) but it caters for the possibility of a program to 'fail', in which case there are no resulting states.

Example 1 We use the following example of an epistemic model M (see Figure 1). Consider $Ag = \{1, 2, 3, 4, 5, 6\}$ and $\mathcal{V}ar = \{x, y\}$. Moreover, $V(1) = \{x\}$, $V(2) = \{y\}$, $V(3) = V(4) = V(5) = \{\}$ and $V(6) = \{x, y\}$. In the figure, we have given each state a name and the valuation it represents. For instance, $f(z)$ is the valuation that assigns *true* to x and *false* to y. Reflexive or transitive arrows are not drawn, for instance every agent considers s possible given s, and we also have $R_3(w, u)$ and $R_3(v, z)$, for example. Agent 4 knows the value of x, and he knows the value of y if x is *true*, and agent 5 knows the value of y, and also that of x if y is *false*. Agent 6 sees, and hence knows, the value of both variables.

Definition 3 (Truth) *Let $M = \langle W, R, V, f\rangle$ with $W \neq \emptyset$ be an epistemic model, and $\mathsf{M}, \mathsf{w} \in \mathsf{PM}$ a pointed program model. The latter induces a relation between pointed epistemic models, where $(M, w)[\![\mathsf{M}, \mathsf{w}]\!](M', w')$ means that execution of the pointed program model M, w in (M, w) leads to (M', w'): the*

relation $(M, w)[\![\mathsf{M}, \mathsf{w}]\!](M', w')$ *will be defined in Section 3. We define the notion "formula* φ *of the language* $\mathcal{L}$ *is true at* (M, w)*", denoted* $(M, w) \models \varphi$*, recursively as follows:*

$(M, w) \models \top$ *is always the case;*
$(M, w) \models x$ *iff* $f(w)(x) = true$*;*
$(M, w) \models V_i x$ *iff* $x \in V(i)$*;*
$(M, w) \models \neg\varphi$ *iff not* $(M, w) \models \varphi$*;*
$(M, w) \models \varphi \wedge \psi$ *iff* $(M, w) \models \varphi$ *and* $(M, w) \models \psi$*;*
$(M, w) \models K_i\varphi$ *iff* wR_iu *implies* $(M, u) \models \varphi$*;*
$(M, w) \models [\mathsf{M}, \mathsf{w}]\varphi$ *iff* $(M, w)[\![\mathsf{M}, \mathsf{w}]\!](M', w')$ *implies* $(M', w') \models \varphi$*.*

We say that a formula φ in $\mathcal{L}$ is *valid in* M if it is true at every (M, w) with $w \in M$, and φ is *valid* if it is valid in every model M. Of course these notions of validity are relative to the set PM that we will shortly specify.

Definition 4 (Model Equivalence) *Let* $M = \langle W, R, V, f\rangle$ *and* $M' = \langle W', R', V', f'\rangle$ *be two epistemic models, with* $w \in M$ *and* $w' \in M'$*. Then the pointed epistemic models* (M, w) *and* (M', w') *are* equivalent *(denoted* $M, w \equiv M', w'$*) if for all* $\varphi \in \mathcal{L}$*:* $M, w \models \varphi$ *iff* $M', w' \models \varphi$*. Moreover,* M *and* M' *are* equivalent *(denoted* $M \equiv M'$*) if for all* $\varphi \in \mathcal{L}$*:* $M \models \varphi$ *iff* $M' \models \varphi$*.*

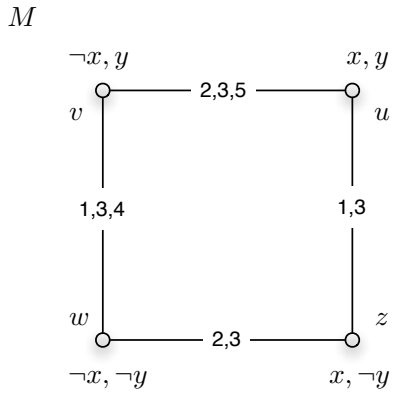

Figure 1: $M = \langle W, R, V, f\rangle$.

3 Program Models

In Dynamic Epistemic Logic, one way to represent epistemic actions is by using so-called *action models*: see, e.g., [3] and [27, Chapter 6]. Analogously (but nevertheless differently), we will define *program models* M, where the pair (M, w)

represents a program point w in program model M. One basic difference is that we define program models as finite sets of pairs $\mathsf{w} = (\varphi_0, X)$, with φ_0 being a consistent propositional formula and $X \subseteq \mathcal{V}ar$ a set of variables. Similarly to actions in action models, the idea behind such a pair (φ_0, X) is that φ_0 represents the precondition for the program point to be executed, and X is the set of variables affected by the point, so they represent in some sense the postcondition of w. We identify points (φ_0, X) and (ψ_0, Y) iff φ_0 and ψ_0 are equivalent, and $X = Y$. More formally, consider the Lindenbaum-Tarski algebra of $\mathcal{L}_0$. Let $\mathcal{A}$ be its carrier set, i.e., the set of equivalence classes $[\varphi]$ based on tautological equivalence. When clear from context we will often write φ instead of $[\varphi]$. We define $\mathcal{A}^- = \mathcal{A} \setminus \{\bot\}$. In words: $\mathcal{A}^-$ represents the set of consistent propositional formulas that are based on $\mathcal{V}ar$, with the constraint that all members are mutually non-equivalent.

Definition 5 (Program Points and Models) *Let $\mathcal{A}^-$ and $\mathcal{V}ar$ be as before.*

- *An element $\mathsf{w} \in \mathcal{A}^- \times \mathcal{P}(\mathcal{V}ar)$ is called a* program point*.*
  A program point $\mathsf{w} = (\varphi, X)$ is called a basic program, if $|X| \leq 1$. A basic program is atomic if either $\varphi = \top$ or $X = \{\}$.
  For any $\mathsf{w} = (\varphi, X)$ we define the projections $pre(\mathsf{w}) = \varphi$, called the precondition for w, and $tgl(\mathsf{w}) = X$, where $\mathsf{tgl}(\mathsf{w})$ are the variables that are toggled by w.

- *Any finite set of program points is called a* program model*.*
  Additionally, given a vision function V, the relation $\approx_i^V \subseteq \mathsf{M} \times \mathsf{M}$ is defined as follows: for all $\mathsf{u}, \mathsf{w} \in \mathsf{M}$:

$$\mathsf{w} \approx_i^V \mathsf{u} \text{ iff } (\mathsf{tgl}(\mathsf{w}) \triangle \mathsf{tgl}(\mathsf{u})) \cap V(i) = \emptyset$$

  The set of all pointed program models M, w, given $\mathcal{L}_0$ and $\mathcal{V}ar$, is denoted by $\mathsf{PM}(\mathcal{L}_0, \mathcal{V}ar)$ or PM, if the parameters are clear from context.

So, given the visible variables $V(i)$ of agent i, he cannot distinguish two program points w and u if the changes they bring about are the same for the variables in $V(i)$. That is, agent i cannot distinguish w and u if for every variable $x \in V(i)$, either x gets changed by both w and u, or else both w and u leave x untouched.

Unlike actions models, program models do not have an indistinguishability relation as part of their definition. This is because our program models provide a so-called *concrete semantics*, in the sense that the accessibility relations $\approx_i^V$ are not abstract, given equivalence relations, but rather, they are exclusively derived from our notion of visibility of the variables – the function V – and the toggle sets tgl. In this sense, our program models are related to arbitrary action models (in the sense of [27]) in the same way as interpreted systems ([8]) are related to general models for knowledge ([15]). Moreover, we want *at most one* program point of the same type in a program model. This is translated as considering propositionally equivalent preconditions to be 'the same' and names for the

program points not being part of the definition (again in contrast with action models or even Kripke frames). The two differences above are essentially what allow us to represent program models as a subset of a Cartesian product. Finally, we insist on the preconditions being (propositionally) *consistent* formulas for technical reasons, in relation to action emulation (Definition 16). This does not affect our results on a conceptual level; if a program model had a program point with an inconsistent precondition, nothing would change if we removed it, as the specific program point could never be applied.

As a final comment on our program models, note that they can be conceived of as pure syntactic objects. This implies that, despite their name, our use of program models M, w in the syntax of our language $\mathcal{L}$ allows us to sidestep (philosophical) discussions relating to the use of semantic objects in the object language: a discussion which action models are frequently prone to (cf. [27, Section 6.1]).

Definition 6 (Model product) *Let $M = \langle W, R, V, f\rangle$ be an epistemic model, and M be a program model. $(M \times \mathsf{M})$ is the epistemic model $M' = \langle W', R', V', f'\rangle$ defined as follows:*

- $W' = \{(w, \mathsf{w}) \mid w \in W, \mathsf{w} \in \mathsf{M} \;\&\; (M, w) \models \mathsf{pre}(\mathsf{w})\}$;
- $(w, \mathsf{w}) R'_i (u, \mathsf{u})$ *iff* $w R_i u$ *and* $\mathsf{w} \approx_i^V \mathsf{u}$;
- $V' = V$;
- $f'((w, \mathsf{w})) = {\updownarrow}\big(\mathsf{tgl}(\mathsf{w})\big)\big(f(w)\big)$.

This definition says that the points of the new epistemic model $M \times \mathsf{M}$ are pairs $(w, \mathsf{w}) \in W \times \mathsf{M}$, but only those pairs for which $\mathsf{pre}(\mathsf{w})$ holds in w. Two such points (w, w) and (u, u) look the same for agent i if the epistemic points w and u look the same for i, and the program points w and u look the same as well, *using the visibility function V from the epistemic model M*. The visibility set stays the same, and the valuation in (w, w) is the one from w, but with the variables in $\mathsf{tgl}(\mathsf{w})$ toggled.

Pending from Truth Definition 3 in Section 2 is the definition of the relation $[\![\mathsf{M}, \mathsf{w}]\!]$. The relation that M, w induces on epistemic pointed models says that $(M, w)[\![\mathsf{M}, \mathsf{w}]\!](M', w')$ iff

$$(M, w) \models \mathsf{pre}(\mathsf{w}) \text{ and } (M', w') = (M \times \mathsf{M}, (w, \mathsf{w}))$$

In words: executing the pointed program (M, w) in the pointed epistemic model (M, w) leads to a new pointed epistemic model (M', w') iff the program (M, w) is executable in (M, w) and the pointed model (M', w') can be obtained as a way of updating (M, w) with the program (M, w).

From the second bullet of Definition 6 it becomes apparent that the vision function V found in the epistemic model is 'borrowed' by the program model being applied. We will need this fact to be reflected in our axiomatisation, and so we will need to describe V with a formula (and not only in the meta-language). Such a characteristic formula can be achieved by a conjunction of literals of the special atoms $V_i x$. Specifically,

$$\chi_V \stackrel{\circ}{=} \bigwedge_{\substack{i\in\mathcal{AG},\\ x\in V(i)}} V_i x \ \ \wedge \bigwedge_{\substack{i\in\mathcal{AG},\\ x\notin V(i)}} \neg V_i x.$$

We list the axioms regarding programs in Table 5 under "Dynamic Component". Of those, ontic change indicates that to calculate the effect of M,w for a propositional formula φ_0, we apply the toggling of the variables collected in $\mathsf{tgl}(\mathsf{w})$ to φ_0. Soundness of all properties in the dynamic component in this table follow from the definition of $[\![\mathsf{M},\mathsf{w}]\!]$ and that of $M \times \mathsf{M}$.

One way to think of pointed program models in the language, is that they are in some sense 'place holders' that signal what kind of changes are happening. To appreciate this statement, it may help to formulate a main result of this paper at this stage, the proof of which is distributed over this section.

Theorem 1 *Every formula ψ is equivalent to a program-free formula ψ'.*

This theorem is proven by 'pushing in' occurrences of $[\mathsf{M},\mathsf{w}]$ in formulas of the form $[\mathsf{M},\mathsf{w}]\varphi$, and then showing that those occurrences can be abandoned when φ is either of the form φ_0 or $V_i x$. The axioms under 'Dynamic Component' of Table 5 indicate how we will achieve this. For instance, we have $[\mathsf{M},\mathsf{w}](\psi_1 \wedge \psi_2) \leftrightarrow ([\mathsf{M},\mathsf{w}]\psi_1 \wedge [\mathsf{M},\mathsf{w}]\psi_2)$. How knowledge operators are dealt with is through the so-called knowledge-reduction property 'program and knowledge' as given in Table 5, and which we display in (kn_red) below. This property is standard in DEL, but because of its importance and the fact that we have to modify it for our setting, we give a proof of its soundness in Lemma 1.

$$[\mathsf{M},\mathsf{w}]K_i\varphi \leftrightarrow \big(\mathsf{pre}(\mathsf{w}) \rightarrow \bigwedge_{V\in\mathcal{V}is} (\chi_V \rightarrow \bigwedge_{\mathsf{w}\approx_i^V \mathsf{u}} K_i[\mathsf{M},\mathsf{u}]\varphi)\big) \qquad \text{(kn\_red)}$$

Lemma 1 *Property* (kn_red) *is valid.*

Proof Let M be an epistemic model with vision function V, $w \in M$ and let M be a program model, with $\mathsf{w} \in \mathsf{M}$. Let $M' = M \times \mathsf{M}$ as defined in Definition 6.

Suppose $M, w \models [\mathsf{M},\mathsf{w}]K_i\varphi \wedge \mathsf{pre}(\mathsf{w})$. For $V' \neq V$ the implications are trivially true, we need only concern ourselves with the single implication involving V. By $M, w \models [\mathsf{M},\mathsf{w}]K_i\varphi$ we have $M', (w,\mathsf{w}) \models K_i\varphi$. So for all (w',w') with $R_i'(w,\mathsf{w})(w',\mathsf{w}')$, we have $M', (w',\mathsf{w}') \models \varphi$. Now let $\mathsf{u} \in \mathsf{M}$ and $v \in M$ be such that $\mathsf{w} \approx_i^V \mathsf{u}$ and $R_i wv$. We need to show that $M, v \models [\mathsf{M},\mathsf{u}]\varphi$. If v does not make $\mathsf{pre}(\mathsf{u})$ true, we are done. So suppose $M, v \models \mathsf{pre}(\mathsf{u})$. Then $(v,\mathsf{u}) \in M'$. Since $R_i wv$ and $\mathsf{w} \approx_i^V \mathsf{v}$, we have $R_i'(w,\mathsf{w})(v,\mathsf{v})$ and hence $M', (v,\mathsf{v}) \models \varphi$, which achieves our aim.

For the converse, suppose $M, w \models \mathsf{pre}(\mathsf{w}) \rightarrow \bigwedge_{V\in\mathcal{V}is}(\chi_V \rightarrow \bigwedge_{\mathsf{w}\approx_i^V \mathsf{u}} K_i[\mathsf{M},\mathsf{u}]\varphi)$. If $M, w \models \neg\mathsf{pre}(\mathsf{w})$, the goal $M, w \models [\mathsf{M},\mathsf{w}]K_i\varphi$ trivially holds. So now suppose $M, w \models \mathsf{pre}(\mathsf{w})$. Given also that $M, w \models \chi_V$, we have $M, w \models \bigwedge_{\mathsf{w}\approx_i^V \mathsf{u}} K_i[\mathsf{M},\mathsf{u}]\varphi$. We want to show that $M', (w,\mathsf{w}) \models K_i\varphi$, so assume $R_i'(w,\mathsf{w})(w',\mathsf{w}')$. By definition of R_i', this means $R_i ww'$ and $\mathsf{w} \approx_i^V \mathsf{w}'$. The first of those statements

implies $M, w' \models [\mathsf{M}, \mathsf{u}]\varphi$ for all u for which $\mathsf{w} \approx_i^V \mathsf{u}$. This in turn implies $M, w' \models [\mathsf{M}, \mathsf{w}']\varphi$, i.e., $M', (w', \mathsf{w}') \models \varphi$.

Note how the dynamic operator gets 'pushed in' when reading the equivalence from left to right. It explains how agent i's knowledge that φ is obtained after the performance of a program M, w: if the program is executable, then, for every program M, u that looks the same for the agent, the agent knows that it will bring about φ.

Another thing to note is that, when 'filtering' for the correct vision function through the use of χ_V we could, instead of going through all vision functions, restrict ourselves only to functions that map agent i to a subset of the variables that appear in the toggle sets of M. For the sake of brevity we chose not to do so.

In epistemic logic with dynamic operators, the principle $[\alpha]K_i\varphi \to K_i[\alpha]\varphi$ is called *no learning*, or *no surprise* (or *no miracles* in [20]): everything an agent will know after the execution of an action α, was already known in advance to be a result of α.

Reversing the direction of *no learning* gives $K_i[\alpha]\varphi \to [\alpha]K_i\varphi$, a property known as *perfect recall*: the agent remembers the effect he anticipates of the action α. We will later in this paper discuss the extent to which our programs satisfy *no learning* and *perfect recall*.

In the next section we identify the place of several meta-logical "programs" in our setting, in fashion similar to dynamic logic.

Definition 7 *The set of programs $\mathcal{P}rog$, is defined (inductively) as follows:*

- *If $\varphi_0 \in \mathcal{L}_0$ is a propositionally consistent formula, then $!\varphi_0 \in \mathcal{P}rog$, called the public announcement of φ_0;*
- *If $x \in \mathcal{V}ar$, then $\updownarrow x \in \mathcal{P}rog$, called the toggle of the value of x;*
- *If $x \in \mathcal{V}ar$, and $\varphi_0 \in \mathcal{L}_0$, then $x := \varphi_0 \in \mathcal{P}rog$, called the assignment of x to the truth value of φ_0;*
- *If $\pi_1, \pi_2 \in \mathcal{P}rog$, then $\pi_1 \cup \pi_2 \in \mathcal{P}rog$, called the non-deterministic choice between π_1 and π_2.*
- *If $\pi_1, \pi_2 \in \mathcal{P}rog$, then $\pi_1; \pi_2 \in \mathcal{P}rog$, called the sequential composition of π_1 and π_2.*

In addition to the usual sequential composition (";") and non-deterministic choice ("$\cup$"), we preoccupy ourselves with public announcement ("!"), as well as an operator $\updownarrow x$ which toggles the value of x (from *false* to *true* or inversely). Public announcement and Toggle are considered the basic constructs for this set of programs as we will shortly see; assignment operations $x := \varphi_0$ can be defined in terms of the other constructs. It is also the case that $\updownarrow x$ could be defined if we instead adopted assignment as our basic construct, so this exchange does not have a serious impact.

For each of the programs in the set $\mathcal{P}rog$ we identify a corresponding program model. Strictly speaking, if $\mathcal{M}$ is the set of all program models, consider a function $I : \mathcal{P}rog \mapsto \mathcal{M}$. $I(\pi) = \mathsf{M}_\pi$ will be the program model for program π. The exact definition for this function is spread over the next subsections. We also introduce the notation $[\pi]\varphi$ for $[\mathsf{M}_\pi]\varphi$.

3.1 Atomic Programs

Recall that atomic programs M, w come in two flavours: we start with those for which $\mathsf{tgl}(\mathsf{w}) = \emptyset$.

Definition 8 (Public Announcement) *The program model for public announcement* $!\varphi_0$ *is the model* $\mathsf{M}_{!\varphi_0} = \{\mathsf{e}\} = \{(\varphi_0, \emptyset)\}$.

By Definition 5, on $\mathsf{M}_{!\varphi_0}$ we have $\approx_i^V = \{(\mathsf{e}, \mathsf{e})\}$ for all $i \in Ag$ and $V \in \mathcal{V}is$.

Example 2 (Public Announcement) See Figure 2 (middle) for the program model for the announcement $!x$. On the right of the figure, we have $M \times \mathsf{M}_{!x}$. This model is as expected: everybody knows that x is *true*, but agents 1 and 3 still do not know the value of y. Also agents 4 and 5 (who do not see any variable) 'remember' the value of both x and y, and agent 6 (who sees all variables) can of course distinguish the two results.

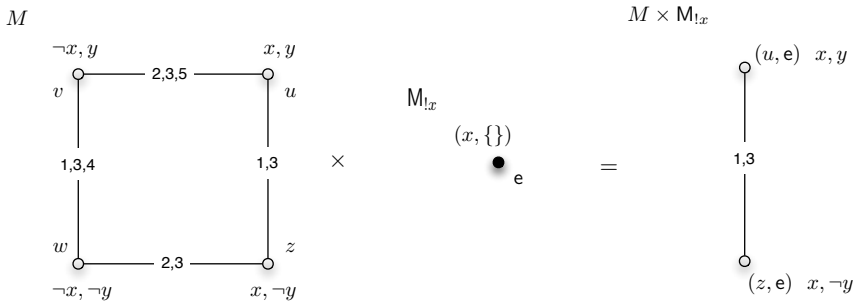

Figure 2: The product $M \times \mathsf{M}_{!x}$.

The knowledge reduction property for announcements is relatively simple:

$$[\mathsf{M}_{!\varphi_0}, \mathsf{e}]K_i\varphi \leftrightarrow (\varphi_0 \to K_i[\mathsf{M}_{!\varphi_0}, \mathsf{e}]\varphi) \qquad \text{(kn\_red(announce))}$$

This property can be derived from (kn_red) by observing that $\mathsf{M}_{!\varphi_0}$ has only one action point e with $\mathsf{e} \approx_i^V \mathsf{e}$ and, moreover, that $\mathsf{pre}(\mathsf{e}) = \varphi_0$. Some derived properties of announcements are given in Table 1.

We have already seen that *no surprise* is not valid for arbitrary programs. However, for pointed program models with a single program point, we immediately derive from (kn_red) that we have something that is very close, i.e.,

Valid properties for announcement ($\varphi_0, \psi_0 \in \mathcal{L}_0,\ \varphi \in \mathcal{L}$)	
$[\mathsf{M}_{!\psi_0}, \mathsf{e}]\varphi \leftrightarrow [!\psi_0]\varphi$	one program point
$[!\psi_0]\varphi_0 \leftrightarrow (\psi_0 \to \varphi_0)$	truthfulness
$[!\psi_0]K_i\varphi \leftrightarrow (\psi_0 \to K_i[!\psi_0]\varphi)$	kn_red(announce)

Table 1: Properties of public announcement.

$[\mathsf{M}, \mathsf{w}]K_i\varphi \to (\mathsf{pre}(\mathsf{w}) \to K_i[\mathsf{M}, \mathsf{w}]\varphi)$. This follows since $\mathsf{w} \approx_i^V \mathsf{w}$, for all i, V and w. Conditioning on $\mathsf{pre}(\mathsf{w})$ is necessary, as we can now show using announcements: suppose x is *false* but i does not know this. Then $[\mathsf{M}_{!x}, \mathsf{e}]K_i\bot$ (since the announcement 'fails'), but obviously we do not have $K_i[\mathsf{M}_{!x}, \mathsf{e}]\bot$.

The reader can verify that *perfect recall* does hold for announcements: from $K_i[!\psi_0]\varphi$ infer $\psi_0 \to K_i[!\psi_0]\varphi$ from which, using kr_red(announce), one concludes $[!\psi_0]K_i\varphi$. More generally, if the program model M has only one point w, then *perfect recall* holds for M, w.

We continue by defining a program model for the other atomic action.

Definition 9 (Simple Toggle) *The program model for the program $\updownarrow x$ that toggles the value of the variable x, is the model $\mathsf{M}_{\updownarrow x} = \{\mathsf{a}\} = \{(\top, \{x\})\}$.*

By Definition 5, we have $\approx_i^V = \{(\mathsf{a}, \mathsf{a})\}$ for all $i \in Ag$ and $V \in \mathcal{V}is$.

Example 3 (Simple Toggle) See Figure 3 (middle) for the program model for $\updownarrow x$. On the right of the figure, we have $M \times \mathsf{M}_{\updownarrow x}$. The properties are straightforward: we have for instance $(M, w) \models [\updownarrow x](K_1 x \wedge K_2\neg y \wedge \neg(K_3 x \vee K_3 y) \wedge K_4 x \wedge K_5\neg y \wedge K_6(x \wedge \neg y))$.

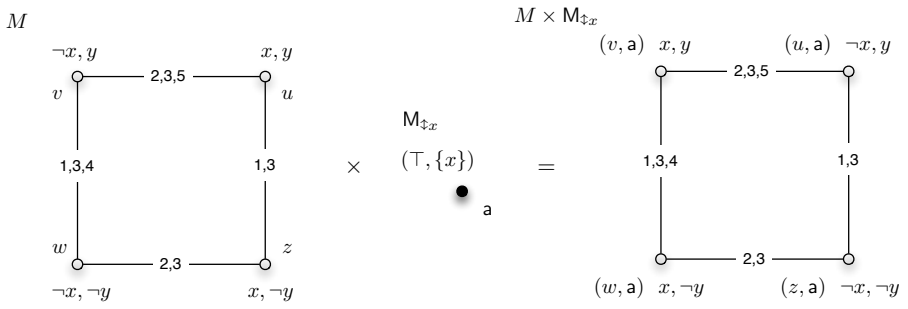

Figure 3: The product $M \times \mathsf{M}_{\updownarrow x}$.

The knowledge reduction property for simple toggle, (kn_red(toggle), see Table 2) is even simpler than that for public announcements. Note that the property does not depend on who sees the toggled variable: both *perfect recall* and *no surprise* hold for simple toggle. An instance of this property (combined

with the 'toggle and objective') is $[\updownarrow \mathsf{x}]K_i x \leftrightarrow K_i \neg x$: agent i knows after toggling of x that x is *true*, if and only if i knows now that x is *false*.

Valid properties for toggle ($\varphi_0 \in \mathcal{L}_0,\ \varphi \in \mathcal{L}$)	
$[\mathsf{M}_{\updownarrow \mathsf{x}}, \mathsf{a}]\varphi \leftrightarrow [\updownarrow \mathsf{x}]\varphi$	one program point
$[\updownarrow \mathsf{x}]\varphi_0 \leftrightarrow \updownarrow(\{x\})(\varphi_0))$	toggle and objective
$[\updownarrow \mathsf{x}]K_i\varphi \leftrightarrow K_i[\updownarrow x]\varphi)$	kn_red(toggle)

Table 2: Properties of simple toggle.

In some sense we are now done: we could proceed by defining program models for non-deterministic choice ($\cup$) and for sequential composition ($;$) and, once those definitions are in place, we could use the equality

$$x := \varphi \stackrel{\circ}{=} \big(!(x \leftrightarrow \varphi) \cup (!(x \leftrightarrow \neg\varphi); \updownarrow x)\big) \tag{1}$$

to define assignments. However, since the definition of program models for choice and sequential composition are a little involved, we first define a program model for assignment, and will then later state that this definition indeed satisfies (1).

3.2 Assignment

We now define the program model for the assignment $x := \varphi$. In fact, as we will later see, $x := \neg x$ and $\updownarrow x$ 'are the same'.

Suppose we chose to model the assignment $x := \varphi$ by a single pointed model M, a. Let (M, w) and (M, u) be two states in an epistemic model. Now suppose that the truth value of φ is different in (M, w) from the value in (M, u). An agent who knows that value will not confuse (M, u) with (M, w), and hence he will also distinguish (w, a) from (u, a). In particular, an agent who knows the truth value of φ before the assignment $x := \varphi$ will know the value of x after it. But now suppose we have an agent i who can see x, but who does not know the value of φ. In particular, suppose for this agent, the states (M, u) and (M, w) look the same. For this agent, the states (w, a) and (u, a) should look different (since the agent sees x, which has a different value in each of them), but, by the definition of product and the fact that $wR_i u$, we get $(w, \mathsf{a}) \approx_i^V (u, \mathsf{a})$, i.e., they look the same. Therefore, we will model the assignment with a program model with two states: one program point in which the value of x stays the same, and one point in which it gets toggled.

Definition 10 (Assignment) *Let $x \in \mathcal{V}ar$ and $\varphi \in \mathcal{L}_0$. The program model for the program $x := \varphi$ is defined (see also Figure 4, left) by stipulating $\mathsf{M}_{x:=\varphi} = \{\mathsf{s} = (x \leftrightarrow \varphi, \{\}), \mathsf{t} = (x \leftrightarrow \neg\varphi, \{x\})\}$.*

It is easy to see that $\mathsf{s} \approx_i^V \mathsf{t}$ iff $x \notin V(i)$. In sum, s denotes that the value of x will stay the same, while t denotes that the value of x as a result of the

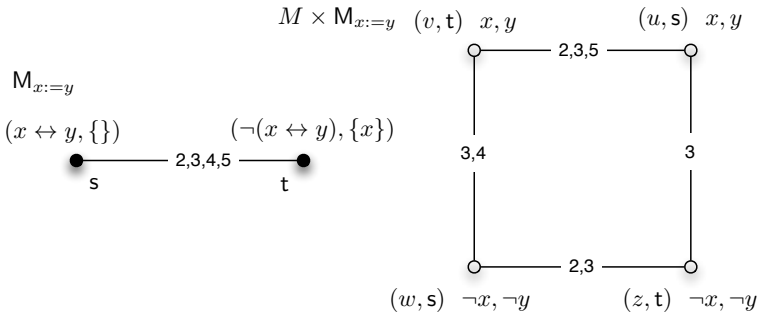

Figure 4: The assignment $\mathsf{M}_{x:=y}$ (left) and the model $M \times \mathsf{M}_{x:=y}$.

assignment, will toggle. Note that the program model for assignment consists of a basic program (toggle) and an atomic one (announcement).

Example 4 (Assignment) Let us consider the assignment $x := y$. Its program model is given in Figure 4 (left). Also, the product $M \times \mathsf{M}_{x:=y}$ of our example epistemic model M with this program model $\mathsf{M}_{x:=y}$ is given at the right of that figure. Note that, as expected, agent 1 (who sees x) comes to learn the value of y. Symmetrically, agent 2 (who sees y) comes to know the value of x.

We display some derived properties of assignment in Table 3. Note that an assignment is basically a choice between two options: either x stays the same or it changes (third line of the table). For agents that see the variable, those options are distinguishable (line 4 and 5), for agents that don't see x, they are not (line 6).

We noted earlier that *perfect recall* holds if the program model only holds one point. We can now use assignments (with models having two points) as an example of failure of *perfect recall*. Note that in Example 4, we have $M, w \models K_4[\mathsf{M}_{x:=y}, \mathsf{s}]\neg x$ (agent 4 knows in w that x is *false*, so he knows that if a program point is executed that keeps the value of x untouched, then afterwards x will still be *false*). However, we do *not* have $M, w \models [\mathsf{M}_{x:=y}, \mathsf{s}]K_4\neg x$ (if an assignment $x := y$ is executed, and it happens to be in a state where both x and y are false, and hence leaves the value of x unchanged, then 4 does not need to know this: for him, y might have the value *true*, in which case the value of x was to be toggled).

Assignments also do not in general satisfy *no learning*. For an agent i who sees x for instance, when $\theta(y) = \mathit{false}$, we have $[\mathsf{x} := \mathsf{y}]K_i\neg y$, but typically not $K_i[\mathsf{x} := \mathsf{y}]\neg y$. Indeed, as an effect of the assignment, the agent has learned the value of y.

Valid properties for assignment ($\varphi_0, \psi_0 \in \mathcal{L}_0,\ \varphi \in \mathcal{L}$)	
$[\mathsf{M}_{\mathsf{x}:=\psi_0}, \mathsf{s}]\varphi_0 \leftrightarrow ((x \leftrightarrow \psi_0) \to \varphi_0))$	no change on s
$[\mathsf{M}_{\mathsf{x}:=\psi_0}, \mathsf{t}]\varphi_0 \leftrightarrow ((x \leftrightarrow \neg\psi_0) \to \updownarrow(\{x\})(\varphi_0))$	toggle on t
$[x := \psi_0]\varphi \leftrightarrow (x \leftrightarrow \psi_0 \wedge [\mathsf{M}_{x:=\psi_0}, \mathsf{s}]\varphi) \vee (x \leftrightarrow \neg\psi_0 \wedge [\mathsf{M}_{x:=\psi_0}, \mathsf{t}]\varphi)$	two choices
$V_i x \to ([x := \psi_0]K_i\varphi \leftrightarrow ((x \leftrightarrow \psi_0) \wedge K_i[\mathsf{M}_{x:=\psi_0}, \mathsf{s}]\varphi) \vee ((x \leftrightarrow \neg\psi_0) \wedge K_i[\mathsf{M}_{x:=\psi_0}, \mathsf{t}]\varphi))$	knowl. & ass. (1)
$\neg V_i x \to ([x := \psi_0]K_i\varphi \leftrightarrow (K_i[\mathsf{M}_{x:=\psi_0}, \mathsf{t}]\varphi \wedge K_i[\mathsf{M}_{x:=\psi_0}, \mathsf{s}]\varphi))$	knowl. & ass. (2)

Table 3: Properties of assignment.

3.3 Non-deterministic choice

The following validity holds both for PDL (ontic change) and DEL (information change):

$$\models ([\pi_1]\varphi_1 \wedge [\pi_2]\varphi_2) \to [\pi_1 \cup \pi_2](\varphi_1 \vee \varphi_2)$$

However, this validity is not appropriate for our framework. Although both $[!x]K_i x$ and $[!\neg x]K_i\neg x$ are validities (regardless of whether $x \in V(i)$), we would not expect $[!x \cup !\neg x](K_i x \vee K_i\neg x)$ to be valid, since the agent simply does not need to know which of the two announcements was in fact executed. So, writing $Kw_i x$ for $K_i x \vee K_i\neg x$ (agent i knows *what* the value of x is, or, equivalently, *whether* x) we have

$$\not\models ([!x]Kw_i x \wedge [!\neg x]Kw_i x) \to [!x \cup !\neg x]Kw_i x$$

Definition 11 (Choice) *Let M_1 and M_2 be two program models. The non-deterministic choice between them is defined as the program model $\mathsf{M}_1 \cup \mathsf{M}_2$. For programs π_1 and π_2, we define $\mathsf{M}_{\pi_1\cup\pi_2} = \mathsf{M}_{\pi_1} \cup \mathsf{M}_{\pi_2}$.*

Example 5 (Choice) Consider first the program $\pi \stackrel{\circ}{=} !x \cup !\neg x$ and the epistemic model M of Example 1. The program model M_π has two actions, e_0 (with precondition $\neg x$) and e_1 (with precondition x). No agent can distinguish e_0 from e_1 (since no variable is affected). It is easy to check that $M \times \mathsf{M}_\pi$ is equivalent to M: indeed, no agent learns anything from the program π.

Secondly, consider the choice between two programs we have considered before: $!x$ and $x := y$ (see the model on the left in Figure 5). Note that we have $M, w \models [!x \cup x := y]K_1(\neg x \wedge \neg y)$. This is as desired: in w, agent 1 knows $\neg x$, so he knows that the announcement $!x$ will not succeed, so the program $x := y$ will be executed. Since in the resulting state, 1 can see that x is *false*, he knows that y is *false*. We also have $M, z \models [!x \cup x := y]K_1(\neg x \wedge \neg y)$. This is for similar reasons as the previous case. So we have $M \models (\neg x \wedge \neg y) \to [!x \cup x := y]K_1(\neg x \wedge \neg y)$. For agent 4, note that in M we had $\neg x \leftrightarrow K_4\neg x$. This is no longer true after execution of the program: 4 cannot distinguish (w, s) (which is the result of executing $x := y$ when both x and y are false), from (v, t) (the result of executing $x := y$ when x is *false* and y is *true*). Note that even agent 6, who sees all variables, is not able to distinguish (u, s) from (u, e): in

Valid properties for choice ($\varphi_0 \in \mathcal{L}_0,\ \varphi \in \mathcal{L}$)	
$[\pi_1 \cup \pi_2]\varphi_0 \leftrightarrow ([\pi_1]\varphi_0 \wedge [\pi_2]\varphi_0)$	choice and objective
$[\mathsf{M}_1 \cup \mathsf{M}_2, \mathsf{w}_1]\varphi_0 \leftrightarrow [\mathsf{M}_1, \mathsf{w}_1]\varphi_0$	if $\mathsf{w}_1 \in \mathsf{M}_1$
$[\mathsf{M}_1 \cup \mathsf{M}_2, \mathsf{w}_2]\varphi_0 \leftrightarrow [\mathsf{M}_2, \mathsf{w}_2]\varphi_0$	if $\mathsf{w}_2 \in \mathsf{M}_2$
$[\mathsf{M}_1 \cup \mathsf{M}_2, \mathsf{w}]K_i\varphi \leftrightarrow$ $\big(\mathsf{pre}(\mathsf{w}) \to \bigwedge_{V \in \mathcal{V}is}(\chi_V \to \bigwedge_{\mathsf{w} \approx_i^V \mathsf{u}} K_i[\mathsf{M}_1 \cup \mathsf{M}_2, \mathsf{u}]\varphi)\big)$	kn_red(choice)

Table 4: Properties of choice.

u, when $x \leftrightarrow y \leftrightarrow \top$, the assignment $x := y$ and the announcement that x are indistinguishable.

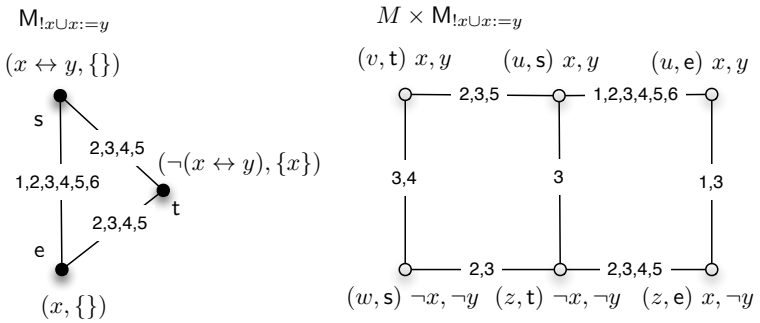

Figure 5: Program model M_π with $\pi \stackrel{\circ}{=} !x \cup x := y$ (left) and $M \times \mathsf{M}_\pi$.

When φ is not propositional there is not necessarily a decomposition of a formula of type $[\mathsf{M}_1 \cup \mathsf{M}_2, \mathsf{w}]\varphi$ in terms of $[\mathsf{M}_1]\varphi$ and $[\mathsf{M}_2]\varphi$. However, the knowledge reduction principle still applies (see Table 4). Note that the actions u for which $\mathsf{w} \approx_i^V \mathsf{u}$, are taken from the full model $\mathsf{M}_1 \cup \mathsf{M}_2$, not only from the model where w is from.

3.4 Sequential composition

We now define program composition of two program models $\mathsf{M}_1, \mathsf{M}_2$.

Definition 12 (Product) *Let two program models M_1 and M_2 be given. The product of $\mathsf{M}_1, \mathsf{M}_2$ is the program model*

$$\mathsf{M}_1 \otimes \mathsf{M}_2 = \{\mathsf{w} \in \mathcal{A}^- \times \mathcal{P}(\mathcal{V}ar) \mid \exists \mathsf{w}_1 \in \mathsf{M}_1, \mathsf{w}_2 \in \mathsf{M}_2 \text{ such that } \mathsf{pre}(\mathsf{w}) = \mathsf{pre}(\mathsf{w}_1) \wedge \updownarrow(\mathsf{tgl}(\mathsf{w}_1))(\mathsf{pre}(\mathsf{w}_2)) \ \&\ \mathsf{tgl}(\mathsf{w}) = \mathsf{tgl}(\mathsf{w}_1) \triangle \mathsf{tgl}(\mathsf{w}_2)\} \tag{2}$$

By denoting a program point by $(\mathsf{w}_1, \mathsf{w}_2)$ *we imply that it exists as a result of the above definition, with* $\mathsf{w}_1, \mathsf{w}_2$ *as the witnesses. For programs* π_1 *and* π_2*, we define* $\mathsf{M}_{\pi_1;\pi_2} = \mathsf{M}_{\pi_1} \otimes \mathsf{M}_{\pi_2}$

This definition is best explained using an example. For this, we use a slightly more involved starting model than before, see Example 6. Informally, the new precondition in $(\mathsf{w}_1, \mathsf{w}_2)$ guarantees that, for a program $\mathsf{M}_1 \otimes \mathsf{M}_2, (\mathsf{w}_1, \mathsf{w}_2)$ to be executable in a given state, the point w_1 needs to be executable now, and w_2 needs to be executable after w_1 is finished. For the program $x := y; z := x$ for instance (so w_1 implements $x := y$ and w_2 implements $z := x$), suppose that we are in a state w where $(x \leftrightarrow \neg y) \wedge (x \leftrightarrow z)$ holds. In order for w_1 to be executable in w, we have $\mathsf{pre}(\mathsf{w}_1) = (x \leftrightarrow \neg y)$, and $\mathsf{tgl}(\mathsf{w}_1) = \{x\}$, and, for w_2 to be executable in w we require $\mathsf{pre}(\mathsf{w}_2) = (x \leftrightarrow z)$ and $\mathsf{tgl}(\mathsf{w}_2) = \{\}$. However, in order for the sequential composition $(\mathsf{w}_1, \mathsf{w}_2)$ to be executable in w, we have to guarantee that $x \leftrightarrow \neg y$ holds, together with $\neg x \leftrightarrow z$: since w_2 expects x and z to have the same truth value, and w_1 will toggle the value of x, before the composition is executed, x and z need to have a different value. In this example, $\mathsf{tgl}(\mathsf{w}_1, \mathsf{w}_2) = \{x, z\}$, which happens to coincide with the union of $\mathsf{tgl}(\mathsf{w}_1)$ and $\mathsf{tgl}(\mathsf{w}_2)$. However, when executing for instance $x := y; x := \neg y$ using two program points, say, w_1 and w_3, the set of variables to be toggled would be empty, which is indeed $\mathsf{tgl}(\mathsf{w}_1) \triangle \mathsf{tgl}(\mathsf{w}_3)$.

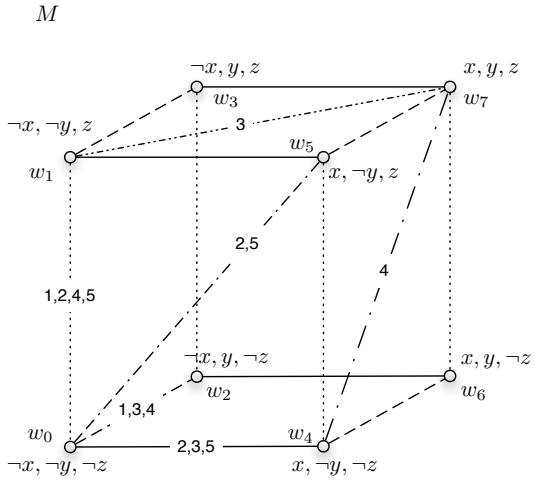

Figure 6: The epistemic model for Example 6. Each pair of parallel lines is labeled with the same agents. Not all lines are drawn in the Figure.

Example 6 (Product) Let us now assume $\mathcal{V}ar = \{x, y, z\}$ and consider the program $\pi \stackrel{\circ}{=} (z := x; x := y; y := z)$, which swaps the values of x and y using a variable z. Let us assume we have five agents: agent 1 sees x, 2 sees y and

3 sees z. Agents 4 and 5, whose relevance will become clear later, do not see any variables. However, agent 4 'happens to know' the value of x, while agent 5 happens to know the value of y. Let $\mathsf{M}_{z:=x}$ have two points s_z ('z stays the same') and t_z ('z toggles'). Similarly for $\mathsf{M}_{x:=y}$ with points s_x and t_x, and model $\mathsf{M}_{y:=z}$ with points s_y and t_y. The individual program models for the three assignments are given in Figure 7.

The states of those models, taken together, give rise to 4 program states (and not 8 as it would be if we did not remove program points with inconsistent preconditions) in the program model for the composition π. The program $((\mathsf{t}_z, \mathsf{s}_x), \mathsf{t}_y)$ for instance indicates that the value of z will change, that of x will stay the same, and that of y will change (note the order of assignments in π). Note however that its precondition is unsatisfiable: indeed, no program that swaps the values of x and y can change the value of y but keep that of x the same, and therefore this program point is not included in the model (we will shortly give an example of how the pre-condition of a product program is computed).

The program model for π is depicted in Figure 8. Let us consider the program $((\mathsf{t}_z, \mathsf{t}_x), \mathsf{t}_y)$, which says that all three values of the variables should change. This would for instance be the case when x is *false*, y and z are *true*. At first sight, one might think that the last assignment $y := z$ would not change the value of y under this valuation, but when performed after $z := x; x := y$ the variable z has of course become false so indeed the assignment $y := z$ will change y's value. This is exactly how the new preconditions in the context of sequential composition are computed.

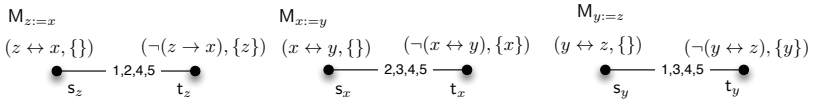

Figure 7: The three assignments $z := x$, $x := y$ and $y := z$.

To stay with the example $((\mathsf{t}_z, \mathsf{t}_x), \mathsf{t}_y)$, according to Definition 12 the program $(\mathsf{t}_z, \mathsf{t}_x)$ has the precondition $\mathsf{pre}(\mathsf{t}_z, \mathsf{t}_x) \stackrel{\circ}{=} (z \leftrightarrow \neg x) \wedge (x \leftrightarrow \neg y)$ and $\mathsf{tgl}(\mathsf{t}_z, \mathsf{t}_x) = \{x, z\}$. Now, to determine the precondition $\mathsf{pre}((\mathsf{t}_z, \mathsf{t}_x), \mathsf{t}_y)$ we obtain $\mathsf{pre}(\mathsf{t}_z, \mathsf{t}_x) \wedge \updownarrow(\mathsf{tgl}(\mathsf{t}_z, \mathsf{t}_x))(\mathsf{pre}(\mathsf{t}_y))$ which (since $\updownarrow(\mathsf{tgl}(\mathsf{t}_z, \mathsf{t}_x))(\mathsf{pre}(\mathsf{t}_y))$ is $\updownarrow(\{x, z\})(y \leftrightarrow \neg z)$ which in turn is $y \leftrightarrow z$) equals $(z \leftrightarrow \neg x) \wedge (x \leftrightarrow \neg y) \wedge (y \leftrightarrow z)$. Indeed, if y initially equals z, and z changes its value, then y will also change its value, during the program π.

The outcome of performing the assignments π to M is given in Figure 9. Where agent 4 knew the value of x in model M, as the result of π he forgets the value of x, but learns the values of y and z. Likewise, while agent 5 knew the value of y in M, in the resulting model he does not remember y, but he has learned the value of x. The agents 1, 2 and 3 all learn the value of z. In fact, 1 and 2 learn the values of all variables, while 3 learns the new values of y and

z. Everybody knows in the new model that $y \leftrightarrow z$. In particular, we have that 3 learns what the old value of x was.

Let us finally mention that there is a program π_{swap} that swaps the values of x and y in such a way that agent 3 does not learn any of their values:

$$!(\neg x \wedge y); x := \top; y := \bot) \cup (!(x \wedge \neg y); x := \bot; y := \top) \cup !x \leftrightarrow y$$

That is, π_{swap} makes a distinction between three mutually exclusive but exhaustive cases:

- x is equivalent to y, in which case, when we want to swap their values, nothing needs to be done;
- currently we have $\neg x \wedge y$, in which case π_{swap} assigns the value *true* to x and *false* to y;
- currently $x \wedge \neg y$ holds, in which case π_{swap} assigns *false* to x and *true* to y.

Note that, e.g., it now holds that $([\pi_{swap}]Kw_i x \leftrightarrow Kw_i y)$: an agent will know the value of x after the swap, only if he knew the value of y before it (likewise, i will know that value of y after the swap only if the value of x is initially known).

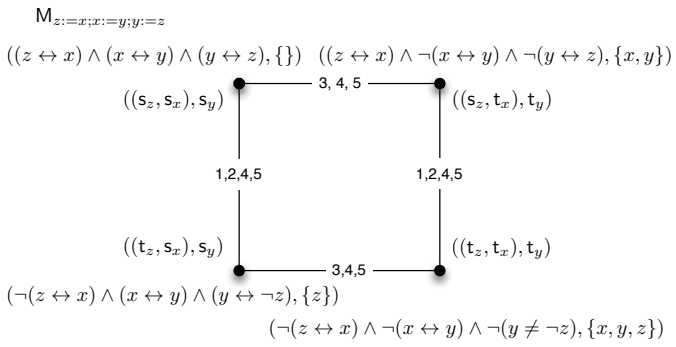

Figure 8: The program model for $z := x; x := y; y := z$. Diagonal connections for the relations $\approx_i^V$ are not shown. For instance, we have $((\mathsf{t}_x, \mathsf{s}_x), \mathsf{s}_y) \approx_4^V ((\mathsf{s}_z, \mathsf{t}_x), \mathsf{t}_y)$.

It is imperative to note that the indistinguishability relation of the composition follows the general definition of program models; it is derived only from the tgl sets of the resulting model (given V). This is in contrast to DEL, where the indistinguishability relations of the action models that are composed come into play. As a result, sequential composition in our setting does not follow the standard behaviour, in the sense that applying two models in turn and applying their composition does *not* always produce equivalent epistemic models (Example 7):

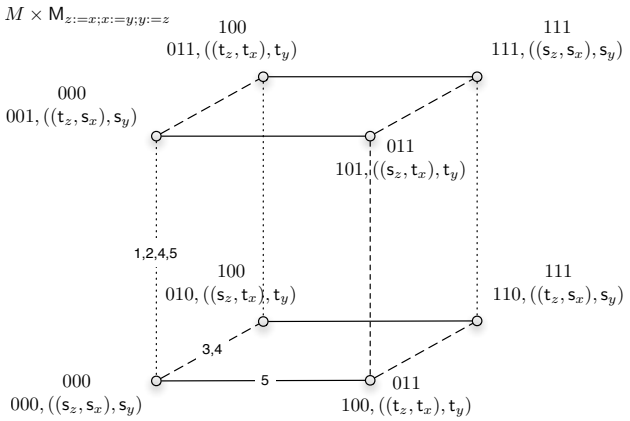

Figure 9: The product $M \times \mathsf{M}_{z:=x;x:=y;y:=z}$.

$$\big((M, w) \times (\mathsf{M}_1, \mathsf{w}_1)\big) \times (\mathsf{M}_2, \mathsf{w}_2) \;\not\equiv\; (M, w) \times \big((\mathsf{M}_1, \mathsf{w}_1) \otimes (\mathsf{M}_2, \mathsf{w}_2)\big)$$

This is perfectly normal on an intuitive level: imagine agent 1 observing an object at all times, while agent 2 leaves and returns some time later, and thus observes only in the end. Agent 1 should be a able to distinguish between the case where the object is moved and then put back, and the case where it was not moved at all, while agent 2 should not.

Example 7 Take an epistemic model with one point w. Suppose $V(1) = \{x\}$. Let $\mathsf{M}_1 = \{\mathsf{u}, \mathsf{v}\} = \{(\top, \{x, y\}), (\top, \{y\})\}$ and $\mathsf{M}_2 = \{\mathsf{w}, \mathsf{z}\} = \{(\top, \{x\}), (\top, \{y\})\}$. Observe that the agent can distinguish between all different states (Def. 5). In $(M \times \mathsf{M}_1) \times \mathsf{M}_2$ the agent can also distinguish between all states (Def. 6), and therefore $Kw_1 y$ is valid. On the other hand, $\mathsf{M}_1 \otimes \mathsf{M}_2 = \{(\top, \{x, y\}), (\top, \{y\}), (\top, \{x\}), (\top, \{\})\}$ and the agent cannot distinguish between $(\mathsf{v}, \mathsf{w}) = (\top, \{x, y\})$ and $(\mathsf{u}, \mathsf{z}) = (\top, \{x\})$, therefore $(M \times (\mathsf{M}_1 \otimes \mathsf{M}_2), (w, \mathsf{v}, \mathsf{w})) \approx_1^V (M \times (\mathsf{M}_1 \otimes \mathsf{M}_2), (w, \mathsf{u}, \mathsf{z}))$. In these two states y has different values so $\neg Kw_1 y$ holds.

3.5 Iteration

In dynamic logic, the operation * (Kleene Star) denotes unlimited repetition: $[\pi^*]\varphi$ is true if φ holds after any arbitrary number of executions of π. It can be used to define the so-called `while` program ('while ψ holds, do π'), as follows: $((?\psi; \pi)^*; ?\neg\psi)$. From a technical point of view, the Kleene star often leads to complications (cf. [11]). In our setting we can deal with the iteration that denotes repetition of the operator ";" in a straightforward way. We remind the reader that this kind of composition is not equivalent to executing actions

serially, as mentioned in the previous subsection. Some additional needed definitions follow.

Definition 13 *Let $\varphi_0, \varphi_1 \in \mathcal{L}_0$. Define $var(\varphi_0) = \{x\}$ if φ_0 is either x or $\neg x$. Define $var(\varphi_0 \wedge \varphi_1) = var(\varphi_0) \cup var(\varphi_1)$. Let w be a program point. Let $var(\mathsf{w}) = var(\mathsf{pre}(\mathsf{w}) \cup \mathsf{tgl}(\mathsf{w}))$. Finally, for a program model M, let $var(\mathsf{M}) = \cup_{\mathsf{w} \in \mathsf{M}} var(\mathsf{w})$.*

Now suppose M is such that $var(\mathsf{M}) = X = \{x_1, \ldots, x_n\}$. Then there are only finitely many program models N with $var(\mathsf{N}) \subseteq X$. This is seen as follows. Let a valuation description over X be a conjunction of the form $l_1 \wedge \cdots \wedge l_n$, with $l_i \in \{x_i, \neg x_i\}$. Then any φ_0 with $var(\varphi_0) = X$ is equivalent to a disjunction of such valuation descriptions. There are 2^n valuation descriptions over X, so there are 2^{2^n} logically different term formulas φ_0 with $var(\varphi_0) = X$. Since a program point is characterised by $\mathsf{pre}(\mathsf{w})$ and $\mathsf{tgl}(\mathsf{w})$, there are at most $2^{2^n} \times 2^n$ different program points over X. Hence there are at most $2^{2^{2^n} \times 2^n}$ different program models M with $var(\mathsf{M}) = X$. Call this number $\#(\mathsf{M})$. Let $\mathsf{M}^1 = \mathsf{M}$ and $\mathsf{M}^k = \mathsf{M}^{k-1}; \mathsf{M}$. We then have:

$$[\mathsf{M}]^*\varphi = [\mathsf{M}]\varphi \wedge [\mathsf{M}^2]\varphi \wedge \ldots [\mathsf{M}^{\#(\mathsf{M})}]\varphi \tag{3}$$

In words: φ is true after an arbitrary number of toggles induced by M iff φ stays true after applying M as often as it is possible to make changes to a model. This shows that iteration is definable in our framework. For this reason, we don't need to deal with iteration any further, and we omit reference to it in our subsequent analysis.

Remark 1 Note that the above definition is not to say that we semantically define M^* as $\mathsf{M}^* = \mathsf{M}^1 \cup \cdots \cup \mathsf{M}^{\#(\mathsf{M})}$. This would indeed not be correct, as the following example shows. Suppose we have an epistemic model M consisting of only one state w, at which x is *true*. In particular, at this state, we have that Kw_ix, i.e., i knows what the value is of x. Let us also assume that $x \notin V(i)$. Now let $\mathsf{M} = \mathsf{M}_{\updownarrow x}$ from Definition 9, the model that simply toggles the value of x. Then $M, w \models [\mathsf{M}]Kw_ix$ and also $M, w \models [\mathsf{M}^2]Kw_ix$: since i knew the value of x, as long as he knows how often this values has been toggled, he will stay aware of that value. However, we do *not* have $M, w \models [\mathsf{M} \cup \mathsf{M}^2]Kw_ix$, since in the program $\mathsf{M} \cup \mathsf{M}^2$, agent i does not know which program point is executed, i.e., he does not know how often x gets toggled. Note that we *do* have $M, w \models [\mathsf{M}]^*Kw_ix$, as desired.

3.6 Inductively Defined Pointed Program Models

We make it explicit that the set of all program models and the set of the program models that can be constructed using the models corresponding to the set of program $\mathcal{P}rog$, coincide.

Definition 14 *We give the set of inductively defined pointed program models, or $\mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$, as follows.*

1. *if* $\varphi_0 \in \mathcal{A}^-$, *then* $\mathsf{M}_{!\varphi_0}, \mathsf{e} \in \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$
2. *If* $x \in \mathcal{V}ar$ *then* $\mathsf{M}_{\updownarrow x}, \mathsf{a} \in \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$
3. *If* $\mathsf{M}, \mathsf{w}, \mathsf{N}, \mathsf{v} \in \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$ *then* $(\mathsf{M} \cup \mathsf{N}, \mathsf{w}), (\mathsf{M} \cup \mathsf{N}, \mathsf{v}) \in \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$
4. *If* $\mathsf{M}, \mathsf{w}, \mathsf{N}, \mathsf{v} \in \mathsf{IPM}$ *and* $\mathsf{pre}(\mathsf{w}) \wedge \updownarrow(\mathsf{tgl}(\mathsf{w}))(\mathsf{pre}(\mathsf{v}))$ *is consistent, then* $(\mathsf{M}, \mathsf{w} \otimes \mathsf{N}, \mathsf{v}) \in \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$.

Theorem 2 $\mathsf{PM}(\mathcal{L}_0, \mathcal{V}ar) = \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$.

Proof It is clear that $\mathsf{PM}(\mathcal{L}_0, \mathcal{V}ar)$ contains $\mathsf{M}_{!\varphi_0}, \mathsf{e}$ and $\mathsf{M}_{\updownarrow x}, \mathsf{a}$, and is closed under non-deterministic choice and 'consistent' sequential composition. This implies that $\mathsf{PM}(\mathcal{L}_0, \mathcal{V}ar) \supseteq \mathsf{IPM}(\mathcal{L}_0, \mathcal{V}ar)$. For $\subseteq$, let $\mathsf{M}, \mathsf{w} \in \mathsf{PM}(\mathcal{L}_0, \mathcal{V}ar)$. Take a program point $\mathsf{u} \in \mathsf{M}$ and let $\mathsf{tgl}(\mathsf{u}) = \{x_1, \ldots, x_n\}$. We have that u is equal to the program $!\mathsf{pre}(\mathsf{u}); \updownarrow x_1; \ldots; \updownarrow x_n$ and so it can obviously be inductively defined. $\mathsf{M} = \bigcup_{\mathsf{u} \in \mathsf{M}} \{\mathsf{u}\}$ thus M itself (and therefore M, w) can be inductively defined.

After this theorem it should be clear that the existence of more than one program points in a model can be attributed to the program's non-deterministic character; the only way to introduce new points is through non-deterministic choice $\cup$. "Vision" then provides a way to resolve uncertainty in case it provides proof of what actually occurred during the computation. On that note we can also elaborate on our statement that non-deterministic choice is given a realistic interpretation. One can alternatively think of semi-public environments that the execution is not being performed publicly, but each agent is given a range of *deterministic* programs (the program points that are of the form $!\mathsf{pre}(\mathsf{u}); \updownarrow x_1; \ldots; \updownarrow x_n$) from which one will be executed. Thus effectively 'hiding' non-deterministic operators behind the agent's ignorance.

3.7 Completeness

Using the validities for our logic of semi-public environments, it is possible to eliminate programs from formulas, thereby proving Theorem 1 (for details on a similar translation – for the logic of action models – we refer the reader to [27]). In other words, we can reduce the logic of semi-public environments to a multi-agent epistemic logic enriched with a notion of vision expressed by the atoms $V_i x$. Starting with the standard $\mathrm{S5}_n$ validities for knowledge modalities K_i, and then by adding two additional axioms for the V_i operators we have a sound and complete axiomatisation for the logic "S5+V". Putting everything together we obtain the following result:

Theorem 3 *The formulas in Table 5 constitute a sound and complete axiomatisation for the logic of semi-public environments.*

Propositional Component	
φ	if φ is a prop. tautology
Epistemic Component	
$V_i x \to (K_i x \vee K_i \neg x)$	seeing implies knowing
$V_i x \to K_j V_i x$	vision is common knowledge
$K_i(\varphi \to \psi) \to (K_i \varphi \to K_i \psi)$	K-axiom
$K_i \varphi \to \varphi$	veridicality (truth axiom)
$K_i \varphi \to K_i K_i \varphi$	positive introspection
$\neg K_i \varphi \to K_i \neg K_i \varphi$	negative introspection
Dynamic Component ($\varphi_0 \in \mathcal{L}_0,\ \varphi, \psi \in \mathcal{L}$)	
$[\mathsf{M}, \mathsf{w}]\varphi_0 \leftrightarrow (\mathsf{pre}(\mathsf{w}) \to \updownarrow(\mathsf{tgl}(\mathsf{w}))(\varphi_0)$	ontic change
$[\mathsf{M}, \mathsf{w}]\, V_i x \leftrightarrow (\mathsf{pre}(\mathsf{w}) \to V_i x)$	vision permanence
$[\mathsf{M}, \mathsf{w}]\neg\varphi \leftrightarrow (\mathsf{pre}(\mathsf{w}) \to \neg[\mathsf{M}, \mathsf{w}]\varphi)$	program and negation
$[\mathsf{M}, \mathsf{w}](\varphi \wedge \psi) \leftrightarrow ([\mathsf{M}, \mathsf{w}]\varphi \wedge [\mathsf{M}, \mathsf{w}]\psi)$	program and conjunction
$[\mathsf{M}, \mathsf{w}]K_i\varphi \leftrightarrow \big(\mathsf{pre}(\mathsf{w}) \to \bigwedge_{V \in \mathcal{V}is}(\chi_V \to \bigwedge_{\mathsf{w} \approx_i^V \mathsf{u}} K_i[\mathsf{M}, \mathsf{u}]\varphi)\big)$	program and knowledge
Rules of Inference	
if $\vdash \varphi$ and $\vdash (\varphi \to \psi)$ then $\vdash \psi$	modus ponens
if $\vdash \varphi$ then $\vdash K_i \varphi$	knowledge-necessitation
if $\vdash \varphi$ then $\vdash [\mathsf{M}, \mathsf{w}]\varphi$	program-necessitation
if $\vdash \psi_1 \leftrightarrow \psi_2$ then $\vdash \varphi[\psi_1/\psi] \leftrightarrow [\psi_2/\psi]$	substitution of equivalents

Table 5: Axiomatisation

Observe that we do not include an axiom to immediately translate formulas of the form $[\mathsf{M}_1, \mathsf{w}_1][\mathsf{M}_2, \mathsf{w}_2]\varphi$ but this can be overcome by starting the translation from the innermost dynamic operator. This is facilitated by the rule of substitution of equivalents (which says that provably equivalent formulas can be substituted for each other within a formula ψ without effecting the truth of ψ —- see for instance [16, Section 5] for more on this rule).

We refer to axiom $V_i x \to K_j V_i x$ as "vision is common knowledge". This terminology may appear to be misleading, given the fact that we do not have operators for common knowledge in our object language. However, the axiom, together with knowledge-necessitation and the K-axiom, guarantees that common knowledge of vision can be obtained in the following sense: any formula of the form $V_i x \to K_{j_1} \dots K_{j_n} V_i x$ is a theorem. (To see a derivation of $V_i x \to K_1 K_2 K_3 V_i x$, note that by the axiom under consideration, we have $V_i x \to K_3 V_i x$, and, by knowledge necessitation $K_2(V_i x \to K_3 V_i x)$ from which, using the K axiom, we obtain $K_2 V_i x \to K_2 K_3 V_i x$. Since we also have $V_i x \to K_2 V_i x$, we then get $V_i x \to K_2 K_3 V_i x$. Applying once more necessitation and the K-axiom for agent 1, we get the desired result).

Finally, note that the models of $S5+V$ are what we intended from the start. Vision of the value of a variable works as a 'lower bound' for the knowledge of its value, and $V_i x$ is either true in all of the possible worlds, or false (in all possible worlds).

4 When are two programs the same?

In modal logic (a good reference is [4]), the notion of *bisimulation* helps to give an answer to the question: when are two pointed models the same? Being bisimilar (written $(M, w) \leftrightarroweq (M', w')$) guarantees that (M, w) and (M', w') satisfy the same static formulas, and on finite models, the converse holds as well. We assume the reader to be familiar with the notion of bisimulation between epistemic models: we state the key clauses without further explanation.

Formally, if $M = \langle W, R, V, f\rangle$ and $M' = \langle W', R', V', f'\rangle$ are two epistemic models, in order for a bisimulation $\mathfrak{R}$ to have $\mathfrak{B}(w, w')$, we require that

Atom $f(w) = f'(w')$ and $\forall i\ V(i) = V'(i)$;

Forth-Bisim If for some $v \in W$ we have $wR_i v$ then for some $v' \in W'$ we have $w'R'_i v'$ and $\mathfrak{B}(v, v')$;

Back-Bisim If for some $v' \in W'$ we have $w'R'_i v'$ then for some $v \in W$ we have $w'R_i v$ and $\mathfrak{B}(v, v')$;

We use $\mathfrak{B} : (M, s) \leftrightarroweq (M', s')$ for the claim that $\mathfrak{B}$ is a bisimulation between M and M' such that $\mathfrak{R}(s, s')$.

Proposition 1 *Suppose* $(M, s) \leftrightarroweq (M', s')$. *Also suppose* M, s *is such that* $M, s \models \mathsf{pre(s)}$. *Then* $(M \times \mathsf{M}, (s, \mathsf{s})) \leftrightarroweq (M' \times \mathsf{M}, (s', \mathsf{s}))$.

Proof The proof of Proposition 1 follows that of [27, Proposition 6.21], and is constructive: assume $\mathfrak{B} : (M, s) \leftrightarroweq (M', s')$ is the witness the bisimulation of the epistemic models. Then consider $\mathfrak{B}'$, defined by $\mathfrak{B}'((u, \mathsf{u}), (u', \mathsf{u}')$ iff both $\mathfrak{B}(u, u')$ and $\mathsf{u} = \mathsf{u}'$. We leave it to the reader to show that $\mathfrak{B}' : (M \times \mathsf{M}, (s, \mathsf{s})) \leftrightarroweq (M' \times \mathsf{M}, (s', \mathsf{s}))$.

Following work by van Eijck *et. al* ([31, 19]) we propose a notion of 'equivalence' for program models. Our presentation is similar to the exposition of [27, Chapter 6, Section 6.1], here, we built on results presented there, and will focus in our proofs on the differences. Note that where [27] talks about action bisimulation and action emulation, in our terminology this will be program bisimulation and program emulation. The idea is that we want to achieve that if two pointed program models emulate, then for every pointed epistemic model (M, w), we have $((M, w) \times (\mathsf{M}, \mathsf{w})) \leftrightarroweq ((M, w) \times (\mathsf{M}', \mathsf{w}'))$, and hence, the two programs (M, w) and $(\mathsf{M}', \mathsf{w}')$ have the same effect (see Proposition 4). Program emulation is weaker than program bisimulation: here, the idea is that having two programs emulate is a sufficient (Theorem 4) and necessary (Theorem 5) condition to make them have the 'same effect' on epistemic models.

Definition 15 (Bisimulation of program models) *Given two pointed program models* (M, w), $(\mathsf{M}', \mathsf{w}')$*, and a vision function* V*, a* program bisimulation *between them is a relation* $\mathfrak{B}_\mathsf{p} \subseteq (\mathsf{M} \times \mathsf{M}')$ *such that* $\mathfrak{B}_\mathsf{p}(\mathsf{w}, \mathsf{w}')$ *and the following three conditions are met for each agent* i *(for arbitrary program points):*

- **Here-Bisim** $_\mathsf{p}$ *If* $\mathfrak{B}_\mathsf{p}(\mathsf{u}, \mathsf{u}')$*, then* $\mathsf{pre}(\mathsf{u}) \leftrightarrow \mathsf{pre}(\mathsf{u}')$ *and* $\mathsf{tgl}(\mathsf{u}) = \mathsf{tgl}(\mathsf{u}')$.
- **Forth-Bisim** $_\mathsf{p}$ *If* $\mathfrak{B}_\mathsf{p}(\mathsf{u}, \mathsf{u}')$ *and* $\mathsf{u} \approx_i^V \mathsf{v}$*, then there is* $\mathsf{v}' \in \mathsf{M}'$ *such that* $\mathfrak{R}(\mathsf{v}, \mathsf{v}')$*;*
- **Back-Bisim** $_\mathsf{p}$ *If* $\mathfrak{B}_\mathsf{p}(\mathsf{u}, \mathsf{u}')$ *and* $\mathsf{u}' \approx_i^V \mathsf{v}'$*, then there is* $\mathsf{v} \in \mathsf{M}$ *such that* $\mathfrak{R}(\mathsf{v}, \mathsf{v}')$*;*

We write $\mathsf{M}, \mathsf{w} \leftrightarroweq_\mathsf{p}^V \mathsf{M}', \mathsf{w}'$ *if there is a program bisimulation* $\mathfrak{B}_\mathsf{p} \subseteq (\mathsf{M} \times \mathsf{M}')$ *based on* V *for which* $\mathfrak{B}_\mathsf{p}(\mathsf{w}, \mathsf{w}')$.

The reader will notice that bisimulation and emulation have a vision function V as a parameter. This is to be expected since we need a vision function V to produce an indistinguishability relation for the program models we are comparing. Also, as we have already stated, we are interested in the effect these programs would have on the same epistemic model, therefore it also makes sense to compare program models using the same V.

Comparing our **Forth-Bisim**$_\mathsf{p}$ clause with that of **Forth-Bisim** above, and with the Forth-clause for action models, the reader would expect

- **Forth-Bisim**$'_\mathsf{p}$ If $\mathfrak{B}_\mathsf{p}(\mathsf{u}, \mathsf{u}')$ and $\mathsf{u} \approx_i^V \mathsf{v}$, then there is $\mathsf{v}' \in \mathsf{M}'$ such that $\mathsf{u}' \approx_i^V \mathsf{v}'$ and $\mathfrak{R}(\mathsf{v}, \mathsf{v}')$;

Indeed, it can be shown that this is equivalent to **Forth-Bisim** $_{\mathsf{p}}$: by **Here-Bisim** $_{\mathsf{p}}$, $\mathfrak{B}_{\mathsf{p}}(\mathsf{u},\mathsf{u}'), \mathsf{u} \approx_i^V \mathsf{v}$ and $\mathfrak{B}_{\mathsf{p}}(\mathsf{v},\mathsf{v}')$, it follows that $\mathsf{u}' \approx_i^V \mathsf{v}'$. A similar remark applies to **Back-Bisim** $_{\mathsf{p}}$. To emphasise this further we provide the following propositions.

For a program model M, a program point w, vision function V and agent i, define $\mathsf{R}_i^{\mathsf{M},V}(\mathsf{w}) = \{\mathsf{u} \in \mathsf{M} \mid \mathsf{w} \approx_i^V \mathsf{u}\}$.

Proposition 2 *Let* $(\mathsf{M},\mathsf{w}),(\mathsf{M}',\mathsf{w}')$ *be two pointed program models. Then:* $\mathsf{M},\mathsf{w} \leftrightarrows_{\mathsf{p}}^V \mathsf{M}',\mathsf{w}'$ *iff for all* $i \in Ag$, $\mathsf{R}_i^{\mathsf{M},V}(\mathsf{w}) = \mathsf{R}_i^{\mathsf{M}',V}(\mathsf{w}')$.

Proposition 3 *Let* (M,w), $(\mathsf{M}',\mathsf{w}')$ *be two pointed program models and* V, Y *two vision functions. If* $\mathsf{M},\mathsf{w} \leftrightarrows_{\mathsf{p}}^V \mathsf{M}',\mathsf{w}'$ *and* $V \subseteq Y$, *then* $\mathsf{M},\mathsf{w} \leftrightarrows_{\mathsf{p}}^Y \mathsf{M}',\mathsf{w}'$

The following proposition captures our main intention for bisimulation.

Proposition 4 *Suppose we have an epistemic pointed model* M, s *with* $M, s \models \mathsf{pre}(\mathsf{s})$ *and vision function* V, *and two pointed program models* M,s *and* M',s' *such that* $\mathsf{M},\mathsf{s} \leftrightarrows_{\mathsf{p}}^V \mathsf{M}',\mathsf{s}'$. *Then*

$$(M \times \mathsf{M},(s,\mathsf{s})) \leftrightarrows (M \times \mathsf{M}',(s,\mathsf{s}'))$$

Proof The proof is similar to that of [27, Proposition 6.23]. Let $\mathfrak{B}_{\mathsf{p}}$ be the program bisimulation between M,s and M',s'. Then $\mathfrak{B}'$ is a witness for the claim we are after:

$$\mathfrak{B}'((u,\mathsf{u})(v,\mathsf{v})) \text{ iff } u = v \text{ and } \mathfrak{B}_{\mathsf{p}}(\mathsf{u},\mathsf{v})$$

It should now be clear why the condition $\mathsf{tgl}(\mathsf{u}) = \mathsf{tgl}(\mathsf{u}')$ in **Here-Bisim**$_{\mathsf{p}}$, which does not occur for action models, is needed: since our programs can change the state, we only want to identify states that make the same changes. The reader should convince herself that our **Here-Bisim**$_{\mathsf{p}}$ clause is necessary to show the **Atom** case in Proposition 4.

The following corollary follows immediately from Propositions 1 and 4.

Corollary 1 *Let* M *be an epistemic model with vision function* V. *If* $(M,s) \leftrightarrows (M',s')$ *and* $(\mathsf{M},\mathsf{s}) \leftrightarrows_{\mathsf{p}}^V (\mathsf{M}',\mathsf{s}')$ *then* $(M \times \mathsf{M},(s,\mathsf{s})) \leftrightarrows (M' \times \mathsf{M}',(s',\mathsf{s}'))$.

We now introduce the notion of program emulation, which is a weaker form of structural similarity compared to that of program bisimulation, but still guarantees bisimilarity of epistemic states, when programs that emulate each other, are executed on it. Roughly, this weakening is obtained as follows. Rather than requiring that two program points (as is program bisimulation) have exactly the same preconditions, for emulation, it is sufficient that one precondition entails the other— as long as the weaker condition is compensated for by a number of alternatives: see the requirements **Forth-Emul** and **Back-Emul** below.

Definition 16 (Emulation) *Given two pointed program models* $(\mathsf{M},\mathsf{w}),(\mathsf{M}',\mathsf{w}')$, *and a vision function* V, *a* program emulation *between them is a relation* $\mathfrak{E} \subseteq (\mathsf{M} \times \mathsf{M}')$ *such that* $\mathfrak{E}(\mathsf{w},\mathsf{w}')$ *and the following three conditions are met for each agent* i *(for arbitrary program points):*

- **Here-Emul** *If* $\mathfrak{E}(\mathsf{u},\mathsf{u}')$*, then* $\mathsf{pre}(\mathsf{u}) \wedge \mathsf{pre}(\mathsf{u}')$ *is consistent and, moreover,* $\mathsf{tgl}(\mathsf{u}) = \mathsf{tgl}(\mathsf{u}')$.
- **Forth-Emul** *If* $\mathfrak{E}(\mathsf{u},\mathsf{u}')$ *and* $\mathsf{u} \approx_i^V \mathsf{v}$*, then there are* $\mathsf{v}'_1, ...\mathsf{v}'_n \in \mathsf{M}'$ *such that for all* $k = 1, ..., n$*,* $\mathfrak{E}(\mathsf{v},\mathsf{v}'_k)$*, and such that* $\mathsf{pre}(\mathsf{v}) \models \mathsf{pre}(\mathsf{v}'_1) \vee \cdots \vee \mathsf{pre}(\mathsf{v}'_k)$ *and* $\mathsf{tgl}(\mathsf{v}) = \mathsf{tgl}(\mathsf{v}'_1) = \cdots = \mathsf{tgl}(\mathsf{v}'_n)$.
- **Back-Emul** *If* $\mathfrak{E}(\mathsf{u},\mathsf{u}')$ *and* $\mathsf{u}' \approx_i^V \mathsf{v}'$*, then there are* $\mathsf{v}_1, ...\mathsf{v}_n \in \mathsf{M}$ *such that for all* $k = 1, ..., n$*,* $\mathfrak{E}(\mathsf{v}_k,\mathsf{v}')$*, and such that* $\mathsf{pre}(\mathsf{v}') \models \mathsf{pre}(\mathsf{v}_1) \vee \cdots \vee \mathsf{pre}(\mathsf{v}_n)$ *and* $\mathsf{tgl}(\mathsf{v}') = \mathsf{tgl}(\mathsf{v}_1) = \cdots = \mathsf{tgl}(\mathsf{v}_n)$.

A total emulation $\mathfrak{E} : \mathsf{M} \rightleftarrows^V \mathsf{M}'$ *is an emulation such that for each* $\mathsf{w} \in \mathsf{M}$ *there is a* $\mathsf{w}' \in \mathsf{M}'$ *with* $\mathfrak{E}(\mathsf{w},\mathsf{w}')$ *and vice versa.*

It should be clear that a program bisimulation is also a program emulation.

Example 8 *Consider the models* $\mathsf{M}_{!\varphi_0 \vee \psi_0} = \{\mathsf{e} = (\varphi_0 \vee \psi_0, \emptyset)\}$ *and* $\mathsf{M}_{!\varphi_0 \cup !\psi_0} = \{\mathsf{e}_1 = (\varphi_0, \emptyset), \mathsf{e}_2 = (\psi_0, \emptyset)\}$*. The reader can easily check* $\mathfrak{E} = \{(e, e_1), (e, e_2)\}$ *is a total emulation.*

The following theorem shows that program emulation guarantees bisimulation of results:

Theorem 4 *Let* M *be an epistemic model with vision function* V*. If* $\mathsf{M} \rightleftarrows^V \mathsf{M}'$ *then* $(M \times \mathsf{M}) \leftrightarrows (M \times \mathsf{M}')$.

Proof The proof of this theorem is similar to that of [27, Proposition 6.30]. The bisimulation that we are after is defined as

$$\mathfrak{B}((w,\mathsf{w}),(w,\mathsf{w}')) \text{ iff } w = v \text{ and } (\mathsf{M},\mathsf{w}) \rightleftarrows^V (\mathsf{M}',\mathsf{w}')$$

We leave it to the reader that $\mathfrak{B}$ indeed is a bisimulation between static epistemic models.

In our setting preconditions are only propositional and, by altering the proof of the related property [6, Proposition 2] accordingly, we get the following extra.

Theorem 5 *Let* M *be an epistemic model with vision function* V*. If* $(M \times \mathsf{M}) \leftrightarrows (M \times \mathsf{M}')$ *then* $\mathsf{M} \rightleftarrows^V \mathsf{M}'$.

It is also natural to consider when two programs are 'the same' directly with respect to what agents can actually reason about – namely, formulas.

Definition 17 *Let us write (in the meta language)* $\pi_1 \sim \pi_2$ *to mean that for all* $\varphi \in \mathcal{L}$*, we have* $\models [\mathsf{M}_{\pi_1}]\varphi \leftrightarrow [\mathsf{M}_{\pi_2}]\varphi$.

This approach has its own interest even outside the context of an agent's epistemic reasoning. Let M be a model and $w \in M$. Applying models M_{π_1} and M_{π_2} will result in worlds of the form (w, u_1) and (w, u_2) with u_1 and u_2 being points of the first and second program model respectively. So we have a partition of the resulting static models based on the first component. The definition then says that the programs are 'equal' if: φ is valid in a part of $M \times \mathsf{M}_{\pi_1}$ iff it is valid in the respective part of $M \times \mathsf{M}_{\pi_2}$. It is therefore evident that this definition is a stronger version of model equivalence.

Theorem 6 *$\pi_1 \sim \pi_2$ iff for all functions $V \in \mathcal{V}is$, $\mathsf{M}_{\pi_1} \rightleftarrows^V \mathsf{M}_{\pi_2}$.*

Proof From left to right: Let V be a vision function and M a model with that vision function. As we pointed out before, we have $(M \times \mathsf{M}_{\pi_1}) \equiv (M \times \mathsf{M}_{\pi_2})$. In our setting all models are finite, so (cf. [5]) we further have $(M \times \mathsf{M}_{\pi_1}) \leftrightarrows (M \times \mathsf{M}_{\pi_2})$. By Theorem 5 we have $\mathsf{M}_{\pi_1} \rightleftarrows^V \mathsf{M}_{\pi_2}$. From right to left: Let $\mathfrak{E}$ be the assumed emulation. By the proof of Theorem 4, we know that there is a bisimulation between $(M \times \mathsf{M}_{\pi_1}, (w_1, \mathsf{u}_1))$ and $(M \times \mathsf{M}_{\pi_2}, (w_2, \mathsf{u}_2))$ if $w_1 = w_2$ and $\mathfrak{E}(\mathsf{u}_1, \mathsf{u}_2)$. Now assume $M, w \models [\mathsf{M}_{\pi_1}]\varphi$ i.e., for every $\mathsf{u} \in \mathsf{M}_{\pi_1}$ such that $M, w \models \mathit{pre}(\mathsf{u})$, it holds that $M \times \mathsf{M}_{\pi_1}, (w, \mathsf{u}) \models \varphi$. Because $\mathfrak{E}$ is a total emulation and because of the fact that bisimilarity implies point-wise equivalence, we also have $M, w \models [\mathsf{M}_{\pi_2}]\varphi$.

Example 9 *We have the following equivalences.*

- $(!\varphi_0 \cup !\psi_0) \sim !(\varphi_0 \vee \psi_0)$
- $\updownarrow x \sim (x := \neg x)$

Remark 2 In our program models, agent i cannot distinguish points u and w if from the variables he sees, those affected by u are the same as those affected by w. This may seem weird, because it might be that u would change x from *false* to *true*, while w does the opposite. However, this is harmless, because those points will have preconditions ($\neg x$ and x, respectively) that i can distinguish.

However, for the Boolean domain one can (and for domains with more than two values, one needs to) model change as follows. Consider $x := \varphi$ again. In the Boolean domain, we take three program points for the program model. One of those three points is similar to s, representing 'stay the same'. Then, we have a state t_1, representing 'change from *false* to *true*', with precondition $(\neg x \wedge \varphi)$ and a program point t_2, representing 'change from *true* to *false*', with precondition $(x \wedge \neg\varphi)$. We replace tgl by a set upd. For instance, we would have $\mathsf{upd}(\mathsf{t}_1) = \{x \mapsto \top\}$ and $\mathsf{upd}(\mathsf{t}_2) = \{x \mapsto \bot\}$, where $\{x \mapsto \top\}\varphi_0$ would then be defined as $\varphi_0[\top/x]$.

5 Discussion

5.1 Related Work

Our work is most closely related to that of van Benthem et al [21] which studies information change combined with 'factual alteration'. This work places

the emphasis on reduction axioms and the underlying system 'is capable of expressing all model-shifting operations with finite action models' [21, page 1]. However, as explained on [21, page 6], an approach where one has regular operations (choice, sequence and iteration) to make structured programs starting from simple actions is not examined. Moreover, there is no notion of visibility in [21].

Our work is also related to 'Dynamic Epistemic Logic with Assignment' [26]. However, there are some important differences. We aim at modelling a computational environment, where *program variables* are first class citizens: they receive values during a commonly known computation, while agents derive their knowledge from the variables they observe and the program that is being executed. The motivation in [26] is more general: its starting point is DEL, to which an assignment of the form $p := \varphi$ is added, that is, some atomic propositions receive a new value. The logic of [26] includes common knowledge and more general update operators than just public announcement. However, there is a price to pay for this generality: [26] gives only one relevant semantic principle, whereas we provide a complete axiomatisation. Moreover, in our framework, the notion of *visibility* (of variables, by agents) plays a central role. This makes it possible to exactly formalise the interaction between assignments, visibility, and knowledge. No such principles are given in [26]. One might argue that $V_i x$ (agent i sees x) in our framework might be mimicked in [26] as $K_i x \vee K_i \neg x$. However, this still does not give the same behaviour in our system: in [26], even if the agent knows the value of x, and he observes the public program $x := y$, if he does not know the value of y, he will 'forget' the value of x! Of course, a formula of the form $V_i x \leftrightarrow K_i x \vee K_i \neg x$ is not valid in our semantics (an agent may know more than is implied by just the variables he sees).

Another line of relevant work is the "Dynamic Logic of Propositional Assignments" ([1]), or DL-PA for short. We claim that for the case of a finite set of variables $\mathcal{V}ar$ we can embed this language into to our framework. We make the comparison firstly based on the language used. In DL-PA the language (let us call it $\mathcal{L}_{DLPA}$) comprises programs $\pi ::= +p \mid -p \mid \pi;\pi \mid \pi \cup \pi \mid \pi^* \mid ?\varphi$ and formulas $\varphi ::= p \mid \top \mid \bot \mid \neg\varphi \mid \varphi \vee \varphi \mid [\pi]\varphi$. Atoms $+p$ (and $-p$ respectively) can be translated in our setting to programs $[p := \top]$ and the rest of the program connectives translate as expected. Another subtle difference is that the tests of DL-PA can include programs, however it is also the case in DL-PA that all formulas are equivalent to some program-free formula, and so we can use this reduct for our version of tests. Regarding semantics, the models of DL-PA are those of propositional logic, i.e., valuations. The translation of this to our setting would be any model $M = \langle W, R, V, f\rangle$, with W the set of all valuations, $f(w) = w$ and for each $i \in \mathcal{AG}$, $V(i) = \emptyset$. The accessibility relation R could be anything again due to the lack epistemic modalities, and for each agent i, $V(i)$ has to be empty so that $\cup$ would indeed be non-deterministic and no agent finds a way around it using his vision. Finally, for $t(\varphi)$ being the translation we mention above, we claim that for every $\varphi \in \mathcal{L}_{DLPA}$, φ is valid in DL-PA iff $t(\varphi)$ is valid in the logic of semi-public environments. We ought to mention that

this claim is not true between DL-PA and PDL (with atomic programs those of propositional assignment).

We should also mention the relationship of our work to the logic AC of action models for DEL [27, Chapter 6]. It should be the case that since DEL and Semi-Public Environments share (almost) the same class of static models, and program models have the capacity for both epistemic and ontic change, all reasoning performed in AC is also possible in semi-public environments. This would be true, were it not for the fact that public announcements in our setting only involve term formulas. We could start by translating action points to program points with empty toggle sets. But action models have the flexibility of having any kind of indistinguishability relation between their action points and by making all program points have empty toggle sets we lose that flexibility. It is then, that one can decide to add 'dummy' variables to the toggle sets, mentioned only therein, and fix the vision functions appropriately, so as to achieve the indistinguishability relation one wants. Regarding the common trait of action and program models, that is, their ability to change the knowledge of an agent: in epistemic models the worlds represent the agent's epistemic alternatives. By using the preconditions of action or program models, one can eliminate alternatives by making worlds 'vanish'. But with program models one can eliminate alternatives also by changing the worlds themselves. The exact relation between the effects of these two approaches, and always within a multi-agent environment, is worth investigating both from a technical and philosophical aspect.

In [14] Levesque gives a version of 'only-knowing' using 'knowing-at-least' and 'knowing-at-most' modalities. $V_i x$ can be thought of as "agent i 'knows-at-least' formula $K_i x \vee K_i \neg x$". This also brings in mind the possibility to implement vision for other formulas as well, so we would have $V_i \varphi$ where φ is not necessarily a (propositional) variable.

In the work of [23] agents can *see* some of the propositional variables (as in our paper), but in addition they can *control* some, i.e., for each variable there is an agent who controls its truth-value. In fact, this work sits in a sequence of papers where the notion of control, and also that of the *transfer* of control is studied ([10, 24, 13]). It is interesting to note here that, no matter whether such an approach includes an epistemic component or not, the dynamics of those systems can be modelled using our program models.

The Situation Calculus studies scenarios like the ones we presented here ([17], and many related papers). Although the language and models in this line of work are similar to ours, more work is needed to establish the precise technical relationships.

It may be tempting to think that there is a connection between our notion of *vision* of variables, and the notion of *awareness*, which, in the context of epistemic logic, dates back to [7] (see also [18] for a contemporary overview). However, there are major differences. For an agent i who is not aware of x, the formula $K_i(x \vee \neg x)$ would typically be false. Compare that to an agent who does not know, and hence does not see the value of x. For such an agent, in our set-up, $K_i(x \vee \neg x)$ would hold. The agent is aware of x, or, put differently, the

agent knows *about* x. Of course, the well-informed agent is he who even knows the value of x (possibly because it is seen by the agent): $(K_i x \vee K_i \neg x)$. In this case, the agent knows the value of x, or, the agent knows *whether* x.

5.2 Conclusions & Future Work

We have provided a sound and complete axiomatisation for a logic of semi-public environments, showing that it is equivalent to epistemic logic enriched with a notion of *vision*. We introduced program models which cater for ontic and epistemic change, which also have a grounded semantics, based on the vision of the agents. Using those program models, we then gave a realistic interpretation of non-deterministic choice, and analysed the non-standard behaviour of sequential composition and iteration within the context of Semi-Public Environments. In contrast to action models, program models can be fully conceived of as syntactic objects, releasing one from the need to justify including them in the object language. Nevertheless, it is possible to take a semantic view of program models, which allowed us to compare them as mathematical structures, in particular, giving sufficient conditions on two program points being 'the same'.

There are many lines of possible future research. A rich vein of possible future work would involve lifting the various restrictions on our model. We might start with the restriction that preconditions are propositional formulas. Another restriction which might be worth softening, is the assumption that it is common knowledge which variables are seen by whom (note that this assumption is also taken in the interpreted systems paradigm: there, it is common knowledge that agents know the value of their local variables, or what the program under execution is). Those assumptions seem related, and removing them may well be a way to reason about *Knowledge-based Programs* [9], where the programs are distributed over the agents, and where it would be possible to branch in a program depending on the knowledge of certain agents. We also ought to consider actions that change vision itself, introduce higher order vision, as well as challenge the automatic connection between vision and knowledge; an agent seeing something might not necessarily mean that he realises it. An aspect of this form of omniscience is present for example in axiom $V_i x \to K_i V_i x$. Van Eijck ([30]) makes such a distinction between actual perception and capability of perception, and includes actions for commonly known changes to vision, unobserved change, and observation of an action witnessed by a group.

Acknowledgements

We would like to thank the reviewers of JLC for their effort and in particular for their significant contribution towards the simplification of this document.

References

[1] P. Balbiani, A. Herzig, and N. Troquard. Dynamic logic of propositional assignments: A well-behaved variant of pdl. In *Proceedings of the 2013 28th Annual ACM/IEEE Symposium on Logic in Computer Science*, LICS '13, pages 143–152, Washington, DC, USA, 2013. IEEE Computer Society.

[2] A. Baltag and L.S. Moss. Logics for epistemic programs. *Synthese*, 139:165–224, 2004. Knowledge, Rationality & Action 1–60.

[3] A. Baltag, L.S. Moss, and S. Solecki. The logic of common knowledge, public announcements, and private suspicions. In I. Gilboa, editor, *Proceedings of the 7th conference on theoretical aspects of rationality and knowledge (TARK 98)*, pages 43–56, 1998.

[4] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge University Press: Cambridge, England, 2001.

[5] P. Blackburn, J.F.A.K. van Benthem, and F. Wolter. *Handbook of Modal Logic*. Studies in logic and practical reasoning. Elsevier, 2007.

[6] J. Van Eijck and J. Ruan. Action emulation. Technical report, CWI, Amsterdam, the Netherlands, 2004. `http://homepages.cwi.nl/~jve/papers/04/ae/ae.pdf`.

[7] R. Fagin and J. Halpern. Belief, awareness, and limited reasoning. *Artificial Intelligence*, 34:39–76, 1988.

[8] R. Fagin, M.Y. Halpern, Y. Moses, and M.Y. Vardi. *Reasoning About Knowledge*. The MIT Press: Cambridge, MA, 1995.

[9] R. Fagin, M.Y. Halpern, Y. Moses, and M.Y. Vardi. Knowledge-based programs. *Distributed Computing*, 10(4):199–225, 1997.

[10] J. Gerbrandy. Logics of propositional control. In *Proceedings of the Fifth International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS-2006)*, pages 193–200, Hakodate, Japan, 2006.

[11] R. Goldblatt. *Logics of Time and Computation (CSLI Lecture Notes Number 7)*. Center for the Study of Language and Information, Ventura Hall, Stanford, CA 94305, 1987. (Distributed by Chicago University Press).

[12] D. Harel, D. Kozen, and J. Tiuryn. *Dynamic Logic*. The MIT Press: Cambridge, MA, 2000.

[13] A. Herzig, T. Lima, E. Lorini, and N. Troquard. A computationally grounded dynamic logic of agency, with an application to legal actions. In Thomas Ågotnes, Jan Broersen, and Dag Elgesem, editors, *Deontic Logic in Computer Science*, volume 7393 of *Lecture Notes in Computer Science*, pages 170–183. Springer Berlin Heidelberg, 2012.

[14] H.J. Levesque. All i know: a study in autoepistemic logic. *Artificial Intelligence*, 42(2–3):263–309, March 1990.

[15] J.-J.Ch. Meyer and W. van der Hoek. *Epistemic Logic for AI and Computer Science*. Cambridge University Press: Cambridge, England, 1995.

[16] G. Mints. *A Short Introduction to Modal Logic*. Number 30 in CSLI Lecture Notes. CSLI Publications, Stanford, USA, 1992.

[17] R.B. Scherl and H.J. Levesque. The frame problem and knowledge-producing actions. *Artificial Intelligence*, 144(1-2):1–39, 2003.

[18] B. Schipper. Awareness and knowledge. In J.Y. Halpern, H. van Ditmarsch, W. van der Hoek, and B. Kooi, editors, *Handbook of Epistemic Logic*. College Publications, 2015. Version at `http://www.econ.ucdavis.edu/faculty/schipper/unaw.htm`.

[19] F. Sietsma and J. van Eijck. Action emulation between canonical models. *Journal of Philosophical Logic*, 42(6):905–925, 2013.

[20] J. van Benthem, J. Gerbrandy, T. Hoshi, and E. Pacuit. Merging frameworks for interaction. *Journal of Philosophical Logic*, 38(5):491–526, 2009.

[21] J. van Benthem, J. van Eijck, and B. Kooi. Logics of communications and change. *Information and Computation*, 204:1620 – 1662, 2006.

[22] W. van der Hoek, P. Iliev, and M. Wooldridge. Knowledge and action in semi-public environments. In J. Lang H. van Ditmarsch and S. Ju, editors, *Logic, Rationality, and Interaction (LORI)*, volume 6953 of *LNCS*, pages 97–110, 2011.

[23] W. van der Hoek, N. Troquard, and M. Wooldridge. Knowledge and control. In K. Tumer, P. Yolum, L. Sonenberg, and P. Stone, editors, *Proc. of 10th Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2011)*, pages 719–726, 2011.

[24] W. van der Hoek, D. Walther, and M. Wooldridge. Reasoning about the transfer of control. *JAIR*, 37:437–477, 2010.

[25] H.P. van Ditmarsch and B.P. Kooi. Semantic results for ontic and epistemic change. In G. Bonanno, W. van der Hoek, and M. Wooldridge, editors, *Logic and the Foundations of Game and Decision Theory*, pages 87–117. Amsterdam University Press, 2008.

[26] H.P. van Ditmarsch, W. van der Hoek, and B. Kooi. Dynamic epistemic logic with assignments. In *AAMAS05*, pages 141–148, 2005.

[27] H.P. van Ditmarsch, W. van der Hoek, and B. Kooi. *Dynamic Epistemic Logic*. Springer-Verlag: Berlin, Germany, 2007.

[28] J. van Eijck. Dynamic epistemic modelling, 2004. Technical Report SEN-E0424.

[29] J. van Eijck. Guarded actions, 2004. Technical Report SEN-E0424.

[30] J. van Eijck. Perception and change in update logic. In Jan Eijck and Rineke Verbrugge, editors, *Games, Actions and Social Software*, pages 119–140. Springer-Verlag, Berlin, Heidelberg, 2012.

[31] J. van Eijck, J. Ruan, and T. Sadzik. Action emulation. *Synthese*, 185(1):131–151, 2012.